Instituto Nacional de Tecnología Industrial

INTI

Ministerio de Industria
Secretaría de industria y Comercio

Control de versiones con *Subversion*

Lic. Renato Cherini
Laboratorio de Testing y Calidad de Software

Sumario

Introducción

Conceptos básicos | Arquitectura de *Subversion*

Ciclo de trabajo básico

Obteniendo contenido | Haciendo cambios | Revisando cambios

| Revisando conflictos | Resolviendo conflictos | Publicando cambios

Temas avanzados

Propiedades | Modo bloqueante

Bifurcación y combinación

¿Por y para qué? | Bifurcaciones | Sincronización | Reintegración

Sumario

Introducción

Conceptos básicos | Arquitectura de *Subversion*

Ciclo de trabajo básico

Obteniendo contenido | Haciendo cambios | Revisando cambios

| Revisando conflictos | Resolviendo conflictos | Publicando cambios

Temas avanzados

Propiedades | Modo bloqueante

Bifurcación y combinación

¿Por y para qué? | Bifurcaciones | Sincronización | Reintegración

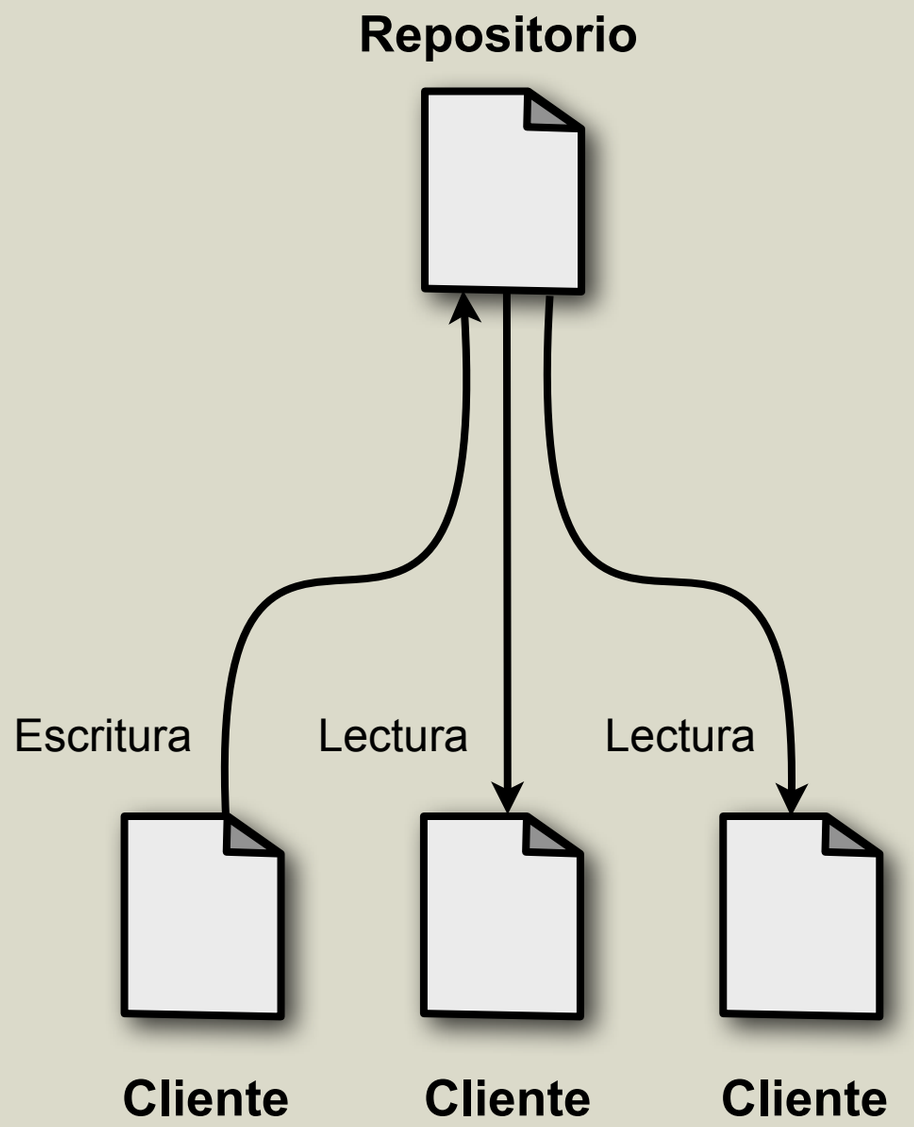
Conceptos básicos

Un *sistema de control de versiones* es un sistema que administra archivos y directorios, y los cambios producidos sobre ellos a lo largo del tiempo, es decir, administra las diferentes versiones incrementales (revisiones) de la información del sistema.

- Permite explorar los cambios introducidos en cada una de las revisiones.
- Facilita acceder a las diferentes revisiones.
- Permite a los usuarios *compartir* la información.

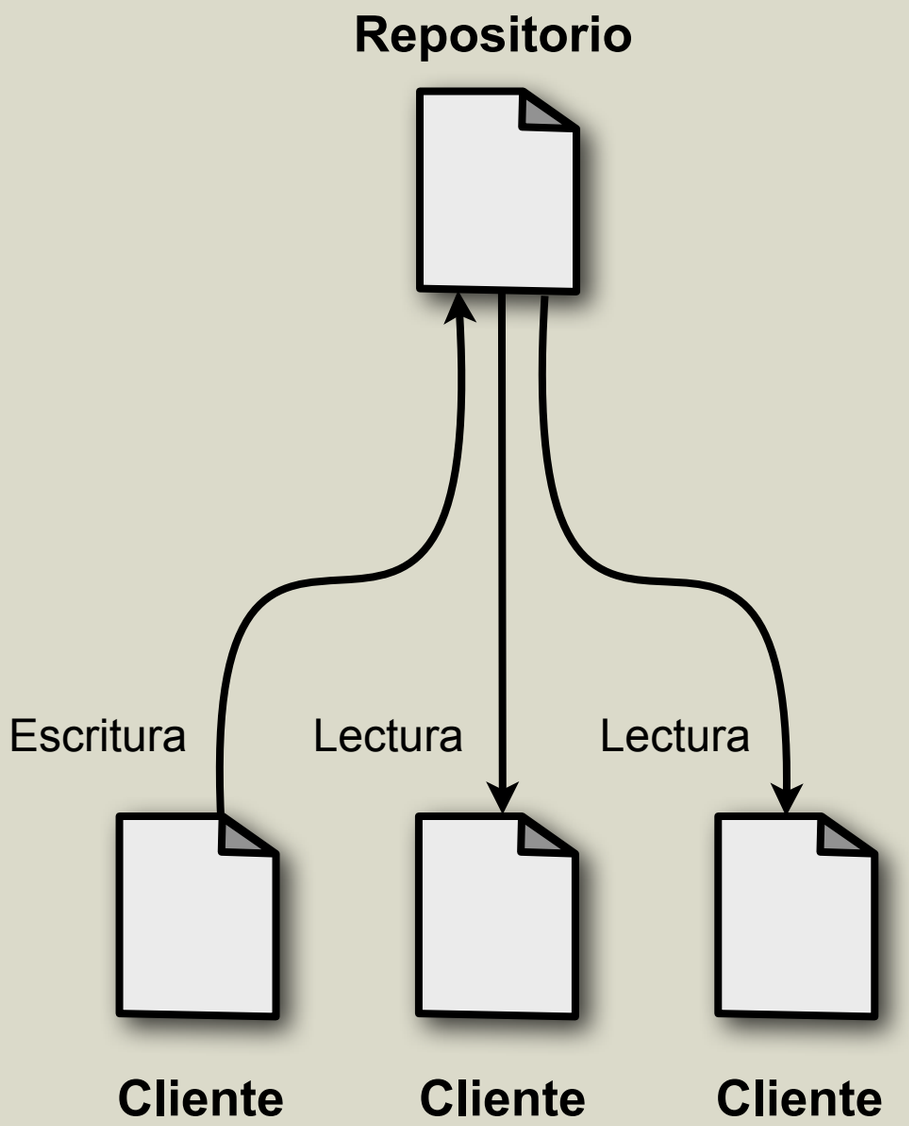
Conceptos básicos: el repositorio

- Es el almacenamiento central de los datos del sistema, usualmente en la forma de un árbol de sistema de archivos.
- Los clientes se conectan para leer o escribir los archivos, permitiendo intercambiar información.



Conceptos básicos: el repositorio

- Es el almacenamiento central de los datos del sistema, usualmente en la forma de un árbol de sistema de archivos.
- Los clientes se conectan para leer o escribir los archivos, permitiendo intercambiar información.
- Cuando se cambian los archivos, el repositorio recuerda *cada versión* de los mismos.
- **Tiene la habilidad de responder a las solicitudes de los estados previos de los archivos y directorios.**



Conceptos básicos: la copia de trabajo local

El valor de un sistema de control de versiones es que almacena todas las revisiones de los archivos y directorios

Conceptos básicos: la copia de trabajo local

El valor de un sistema de control de versiones es que almacena todas las revisiones de los archivos y directorios

Pero el resto del software (editores de código fuente, procesadores de texto, editores de páginas web, etc) solo operan sobre versiones *simples*.

Conceptos básicos: la copia de trabajo local

El valor de un sistema de control de versiones es que almacena todas las revisiones de los archivos y directorios

Pero el resto del software (editores de código fuente, procesadores de texto, editores de páginas web, etc) solo operan sobre versiones *simples*.

Entonces, **¿cómo hace un usuario para interactuar con un repositorio?**

Conceptos básicos: la copia de trabajo local

El valor de un sistema de control de versiones es que almacena todas las revisiones de los archivos y directorios

Pero el resto del software (editores de código fuente, procesadores de texto, editores de páginas web, etc) solo operan sobre versiones *simples*.

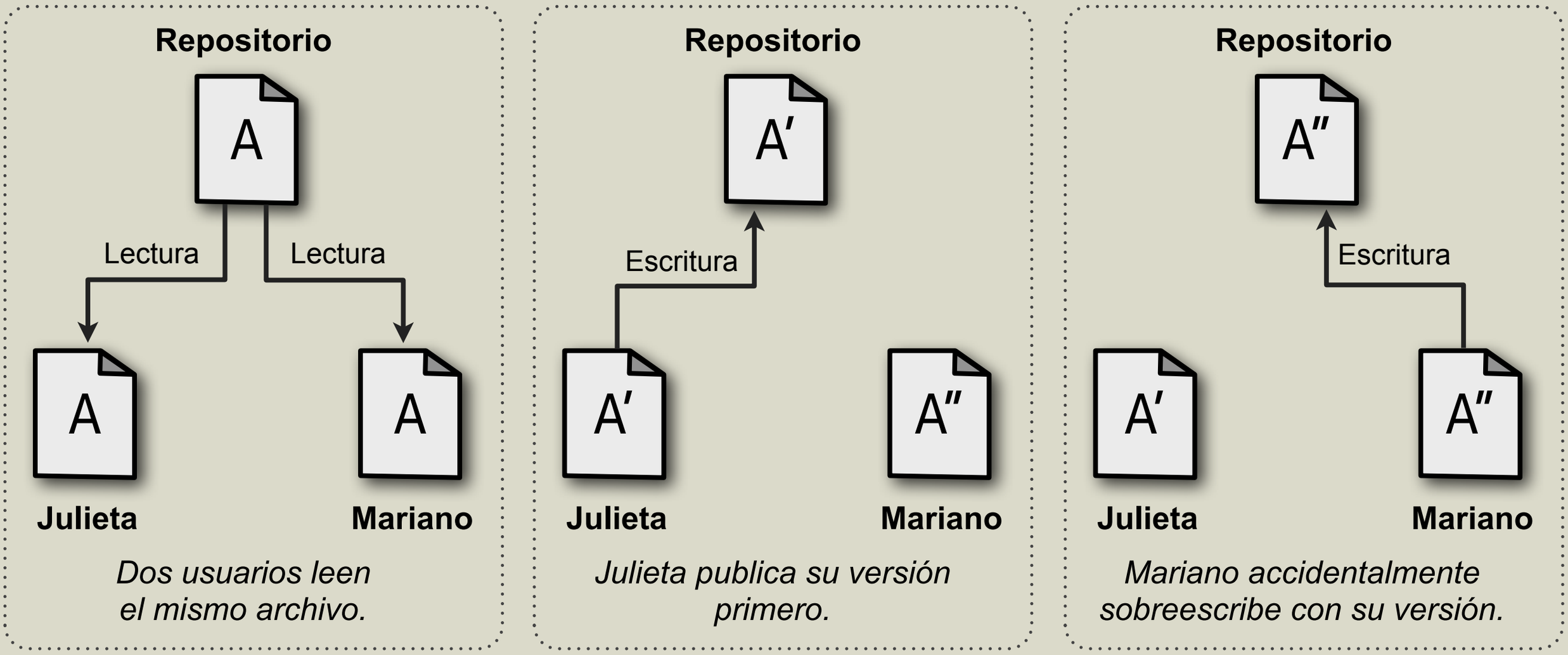
Entonces, **¿cómo hace un usuario para interactuar con un repositorio?**

Con una **copia de trabajo:** copia local de una revisión particular de los datos, sobre la que el usuario es libre de trabajar.

Instituto Nacional de Tecnología Industrial

Conceptos básicos: modelos de versionado

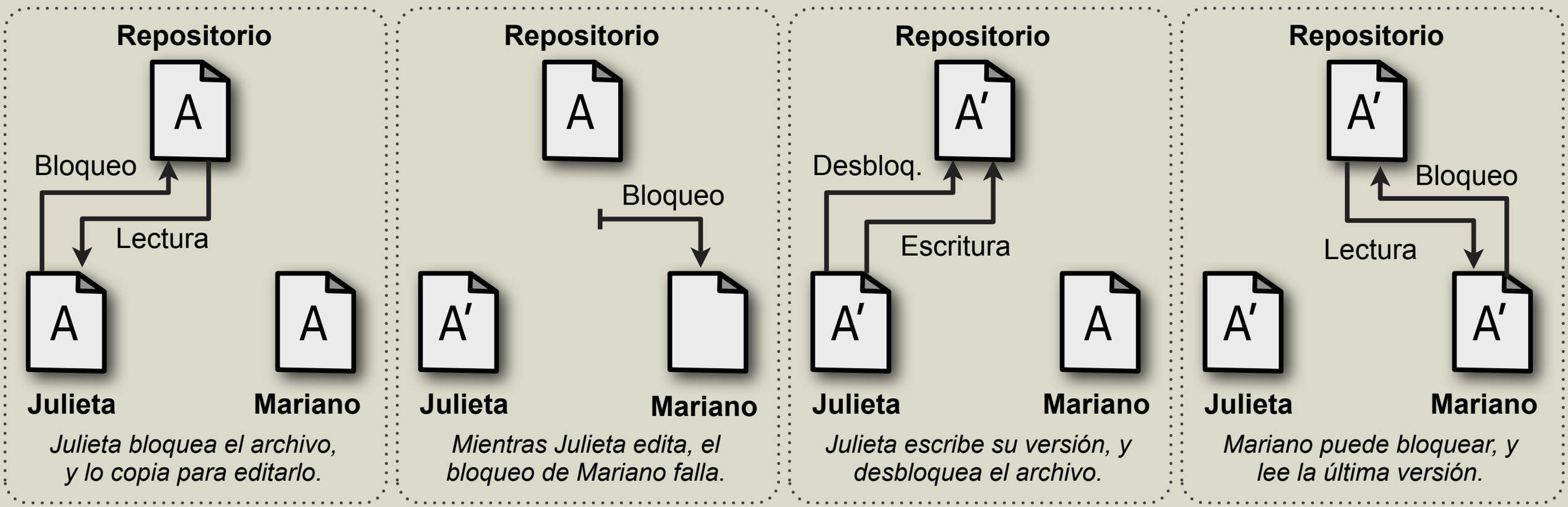
El problema de los archivos compartidos: es muy fácil sobreescribir los cambios de los demás.



Instituto Nacional de Tecnología Industrial

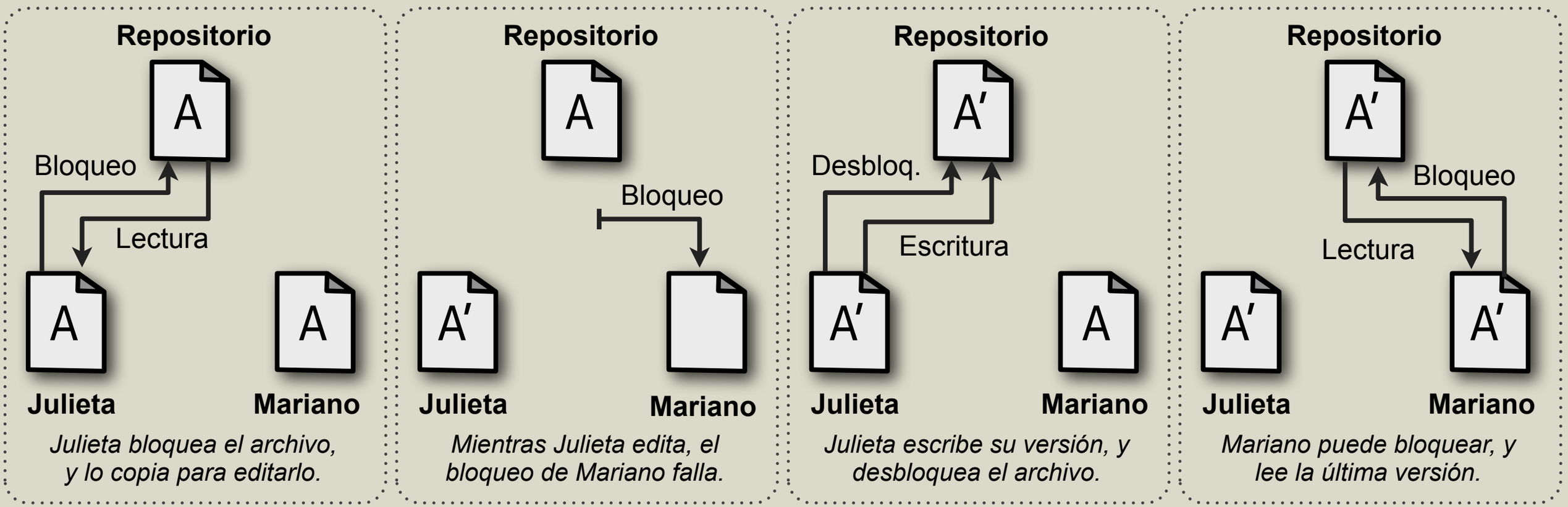
Conceptos básicos: modelos de versionado

La solución bloquear-modificar-desbloquear: solo una persona a la vez puede modificar un archivo del repositorio.



Conceptos básicos: modelos de versionado

La solución bloquear-modificar-desbloquear: solo una persona a la vez puede modificar un archivo del repositorio.



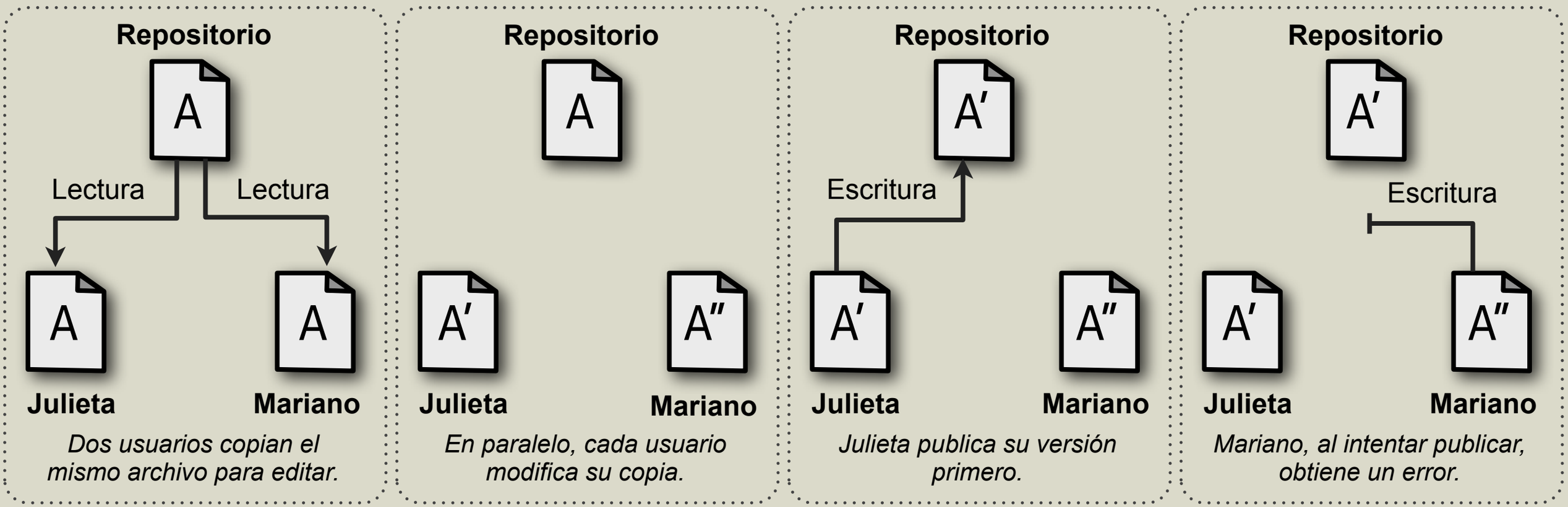
El bloqueo puede causar:

- Problemas administrativos.
- Serialización innecesaria.
- Falsa sensación de seguridad.

Instituto Nacional de Tecnología Industrial

Conceptos básicos: modelos de versionado

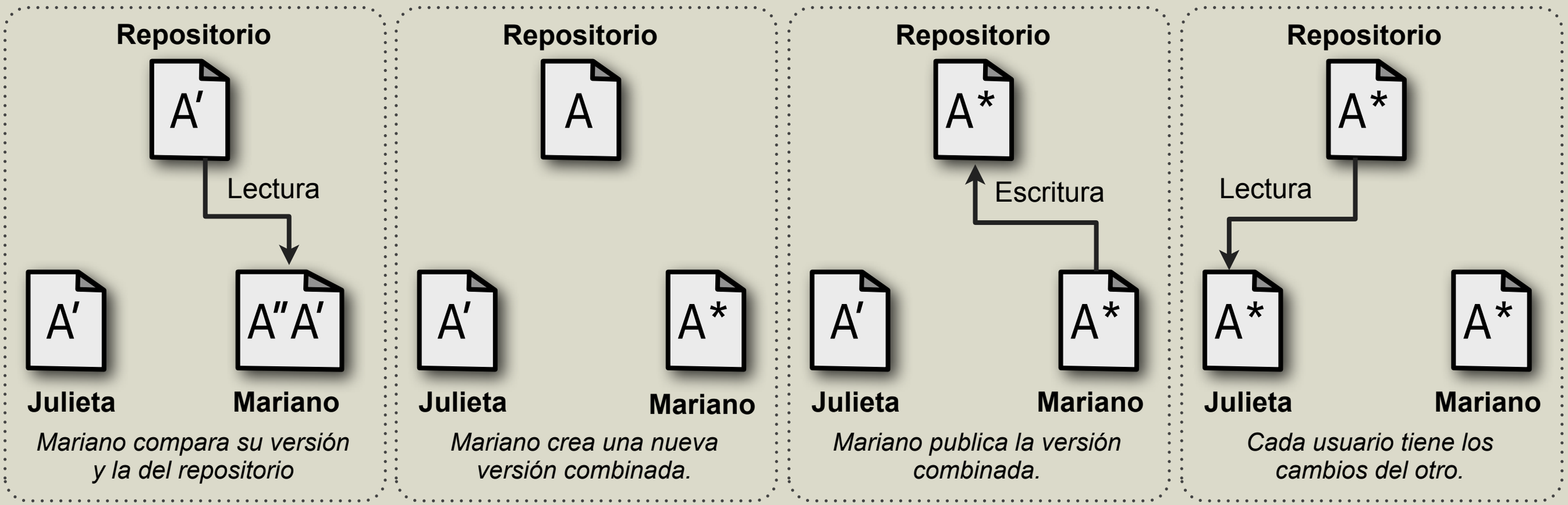
La solución copiar-modificar-combinar: los usuarios crean copias locales privadas, que modifican de forma simultánea e independiente. Luego las copias privadas se combinan en una sola revisión final.



Instituto Nacional de Tecnología Industrial

Conceptos básicos: modelos de versionado

La solución copiar-modificar-combinar: los usuarios crean copias locales privadas, que modifican de forma simultánea e independiente. Luego las copias privadas se combinan en una sola revisión final.



Conceptos básicos: modelos de versionado

La solución copiar-modificar-combinar: los usuarios crean copias locales privadas, que modifican de forma simultánea e independiente. Luego las copias privadas se combinan en una sola revisión final.

- Si dos o mas modificaciones se solapan, se genera un conflicto.
- Cuando existe un conflicto, un usuario capaz de comprender las modificaciones y tomar las decisiones mas acertadas debe corregirlo manualmente.
- Aunque pueda parecer caótico, los conflictos son poco frecuentes.
- Esta solución favorece una buena comunicación, que a su vez reduce tanto los conflictos sintácticos, como los conflictos semánticos.

Conceptos básicos: modelos de versionado

La solución copiar-modificar-combinar: los usuarios crean copias locales privadas, que modifican de forma simultánea e independiente. Luego las copias privadas se combinan en una sola revisión final.

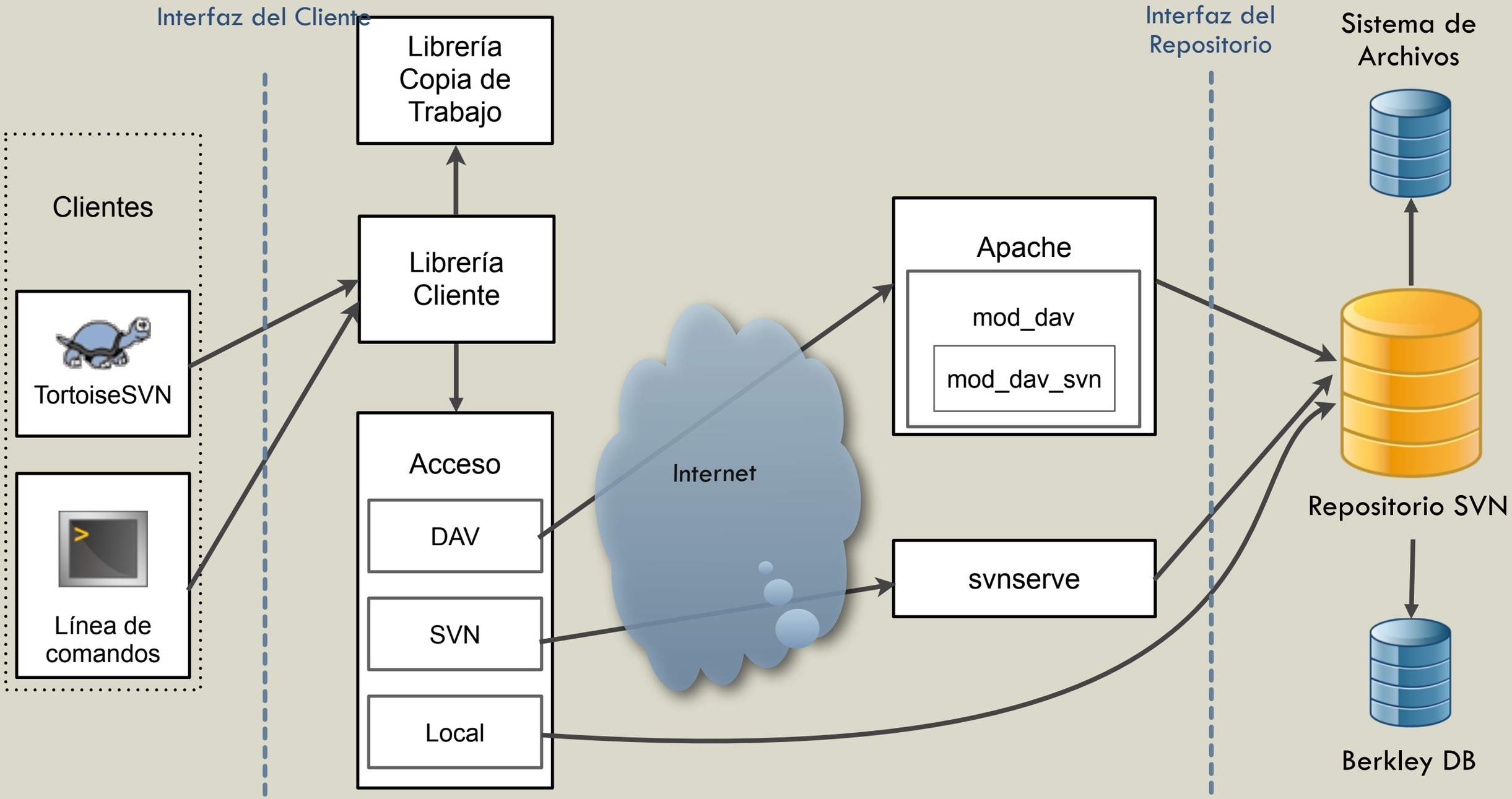
- Si dos o mas modificaciones se solapan, se genera un conflicto.
- Cuando existe un conflicto, un usuario capaz de comprender las modificaciones y tomar las decisiones mas acertadas debe corregirlo manualmente.
- Aunque pueda parecer caótico, los conflictos son poco frecuentes.
- Esta solución favorece una buena comunicación, que a su vez reduce tanto los conflictos sintácticos como los conflictos semánticos.

El bloqueo es necesario en algunas circunstancias

- Este modelo se basa en la suposición de que los archivos son combinables (usualmente archivos de texto).
- Cuando los archivos son binarios, es necesario modificarlos de forma secuencial, para evitar perder el tiempo en cambios que se desecharán.

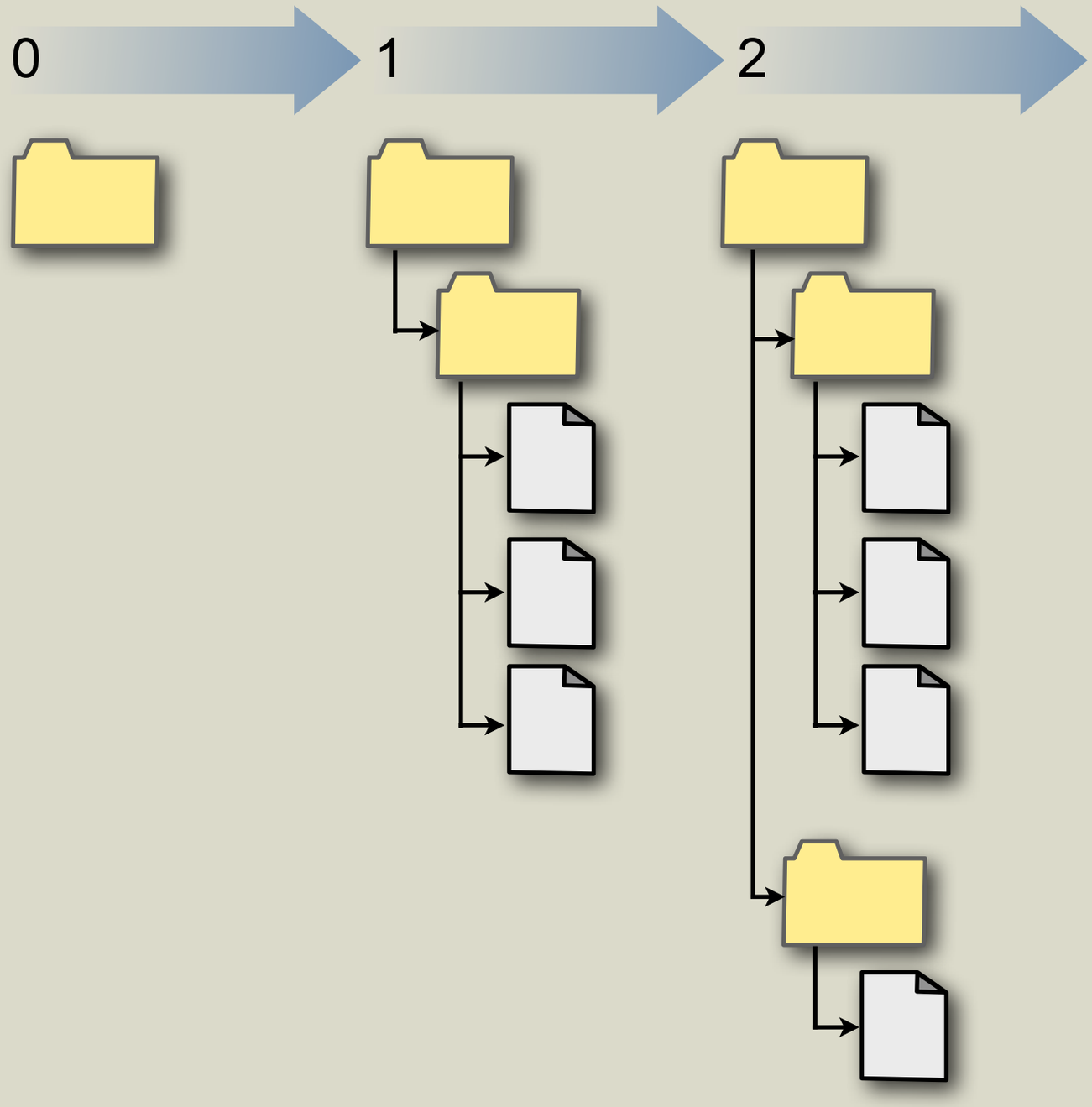
Instituto Nacional de Tecnología Industrial

Arquitectura de SVN



Arquitectura de SVN: revisiones

- Los clientes comunican los cambios a un numero de archivos y directorios de forma **atómica**.
- Cuando se acepta un cambio, el repositorio crea un nuevo estado del sistema de archivos (revisión), asignándole un número único.
- Las revisiones son globales.



Arquitectura de SVN: acceso al repositorio

Los clientes identifican los archivos y directorios versionados en un repositorio SVN utilizando una URL:

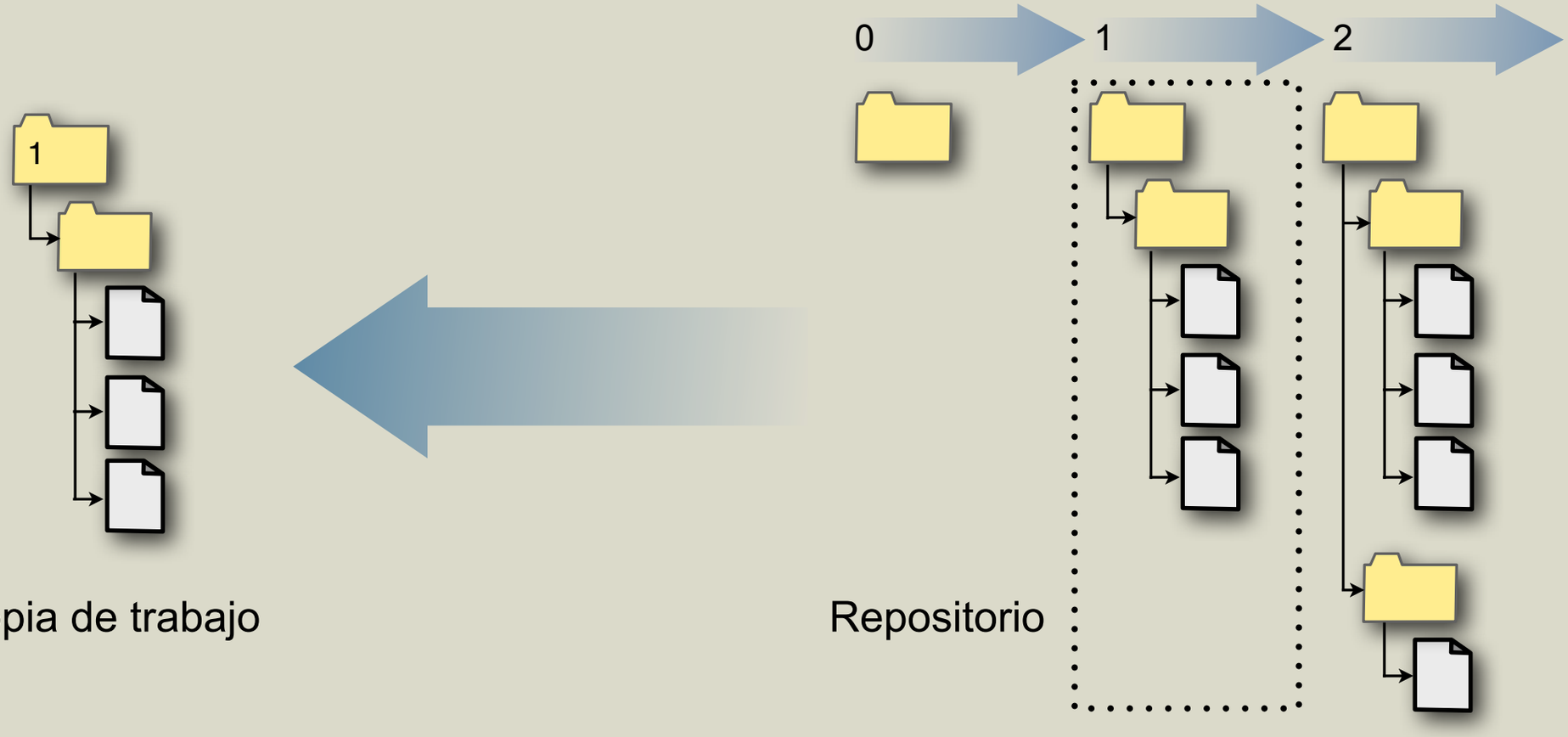
- `http://svn.ejemplo.com.ar/svn/proyecto`
- `http://svn.ejemplo.com.ar:9943/repo`

Esquema	Método de acceso
`file://`	Acceso directo (en el disco duro loca)
`http://`	Acceso Web-DAV en un servidor Apache con soporte SVN
`https://`	Igual que el anterior pero con cifrado SSL
`svn://`	Acceso con diferentes protocolos en un servidor **svnserver**
`svn+ssh://`	Igual que el anterior pero a través de un túnel SSH

Instituto Nacional de Tecnología Industrial

Arquitectura de SVN: copia de trabajo

- Es un árbol de directorios y archivos corriente.
- Es privada: no se incorporan los cambios de terceros, ni se publican los cambios propios salvo que se indique explícitamente.
- Se incluye un *directorio administrativo* para almacenar datos de la revisión.



Arquitectura de SVN: copia de trabajo

- Es un árbol de directorios y archivos corriente.
- Es privada: no se incorporan los cambios de terceros, ni se publican los cambios propios salvo que se indique explícitamente.
- Se incluye un *directorio administrativo* para almacenar datos de la revisión:
  - Revisión en la que se basa el archivo.
  - *Timestamp* cuando se copió el archivo desde el repositorio.

Estado	Descripción
Inmutado, corriente	El archivo no fue modificado localmente, ni se publicaron cambios en el repositorio
Modificado, corriente	El archivo está modificado localmente, pero no se publicaron cambios
Inmutado, antiguo	El archivo no fue modificado localmente, pero publicaron cambios en el repositorio
Modificado, antiguo	El archivo está modificado localmente, y se publicaron cambios en el repositorio

Arquitectura de SVN: copia de trabajo

- Es un árbol de directorios y archivos corriente.
- Es privada: no se incorporan los cambios de terceros, ni se publican los cambios propios salvo que se indique explícitamente.
- Se incluye un *directorio administrativo* para almacenar datos de la revisión.
- Las acciones que publican modificaciones hacia el repositorio y actualizan cambios desde el repositorio son independientes, dando lugar a revisiones mixtas.
    - Las revisiones mixtas son **normales**: se producen por publicar cambios locales producidos en algunos componentes.
    - Las revisiones mixtas son **útiles**: actualizar a revisiones antiguas es útil para *testing* y corrección de defectos.
    - Las revisiones mixtas **tienen limitaciones**: algunas operaciones podrían dar lugar a inconsistencias en el repositorio.

Sumario

Introducción

Conceptos básicos | Arquitectura de *Subversion*

Ciclo de trabajo básico

Obteniendo contenido | Haciendo cambios | Revisando cambios

| Revisando conflictos | Resolviendo conflictos | Publicando cambios

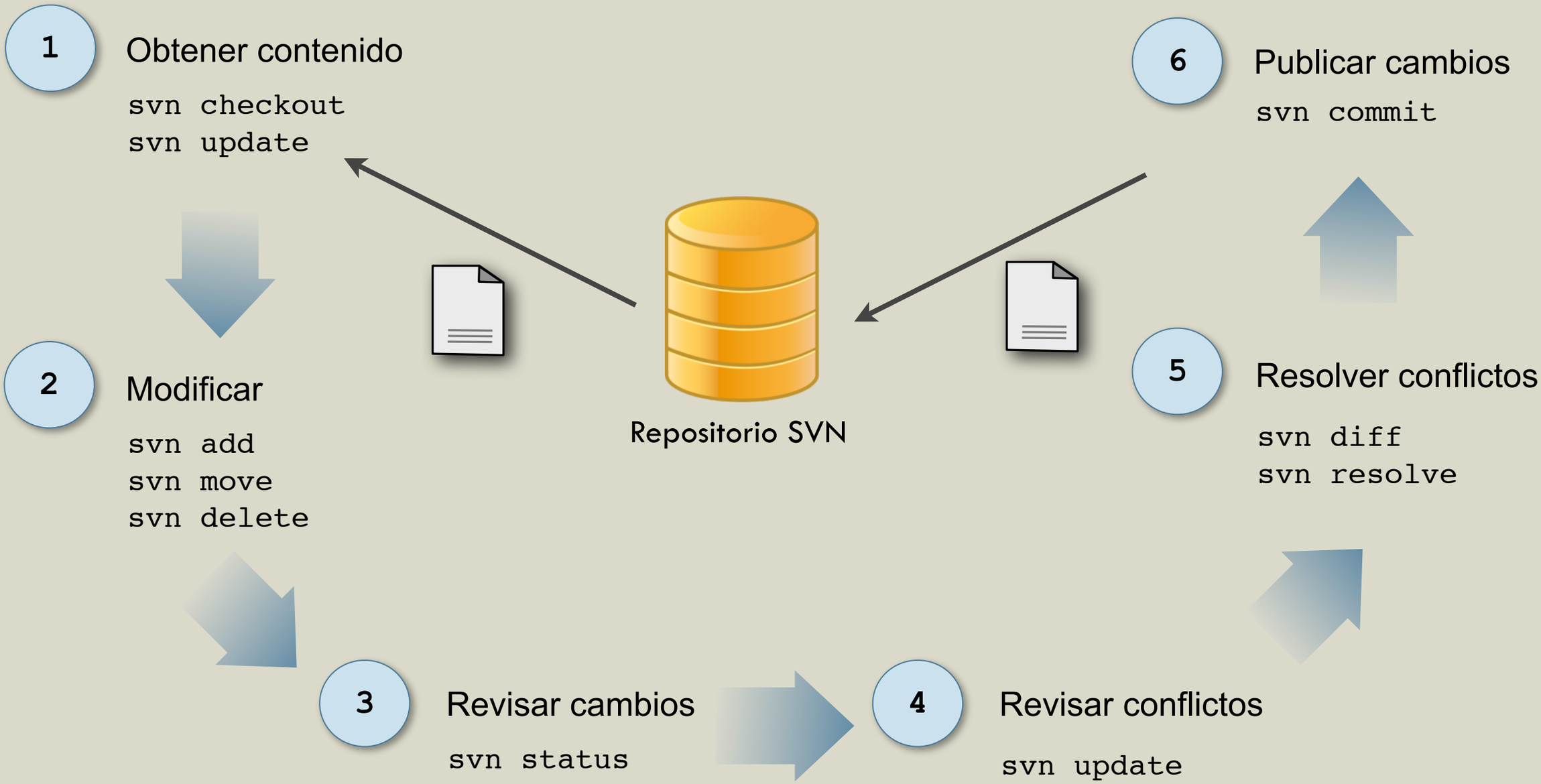
Temas avanzados

Propiedades | Modo bloqueante

Bifurcación y combinación

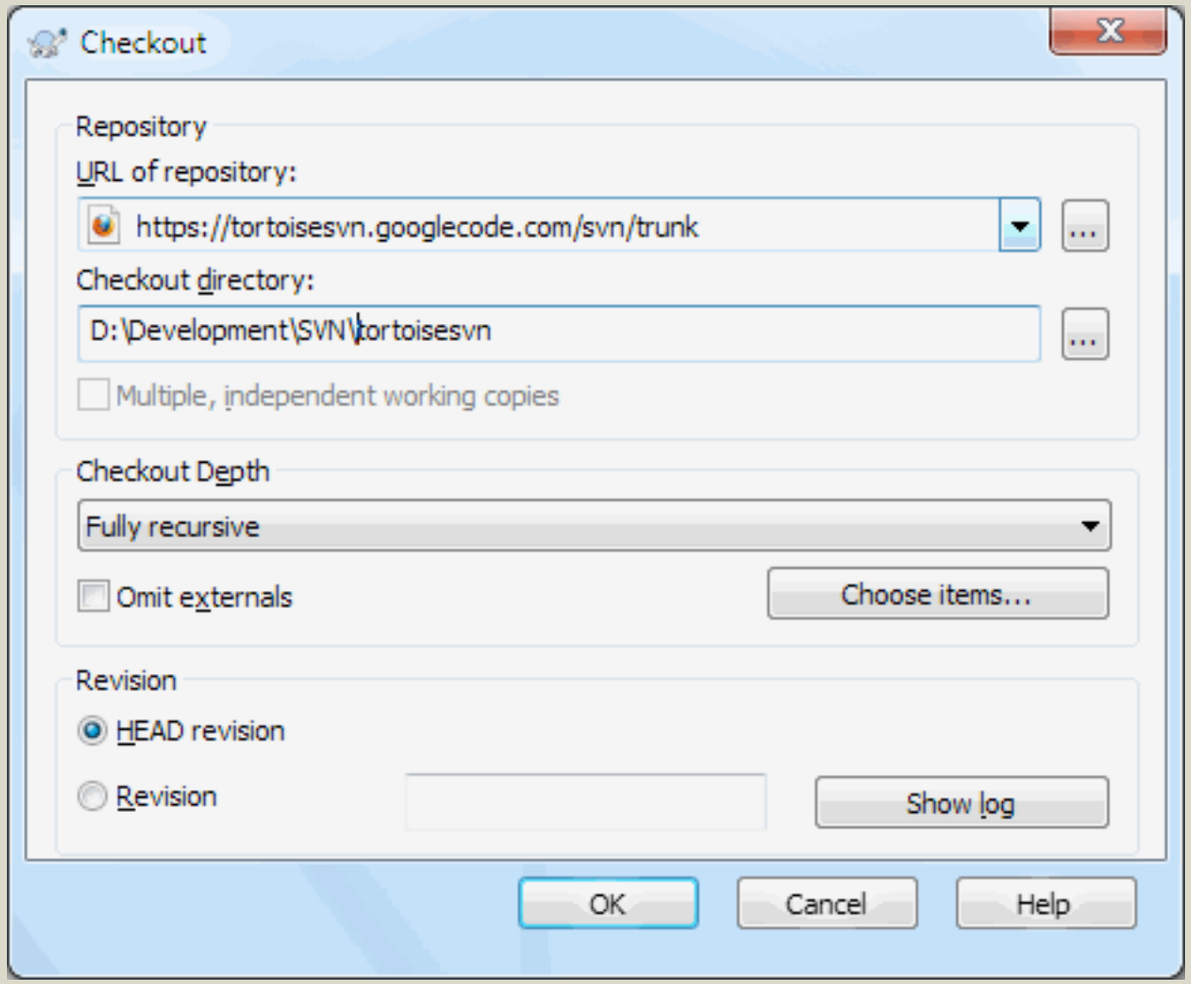
¿Por y para qué? | Bifurcaciones | Sincronización | Reintegración

Ciclo de trabajo



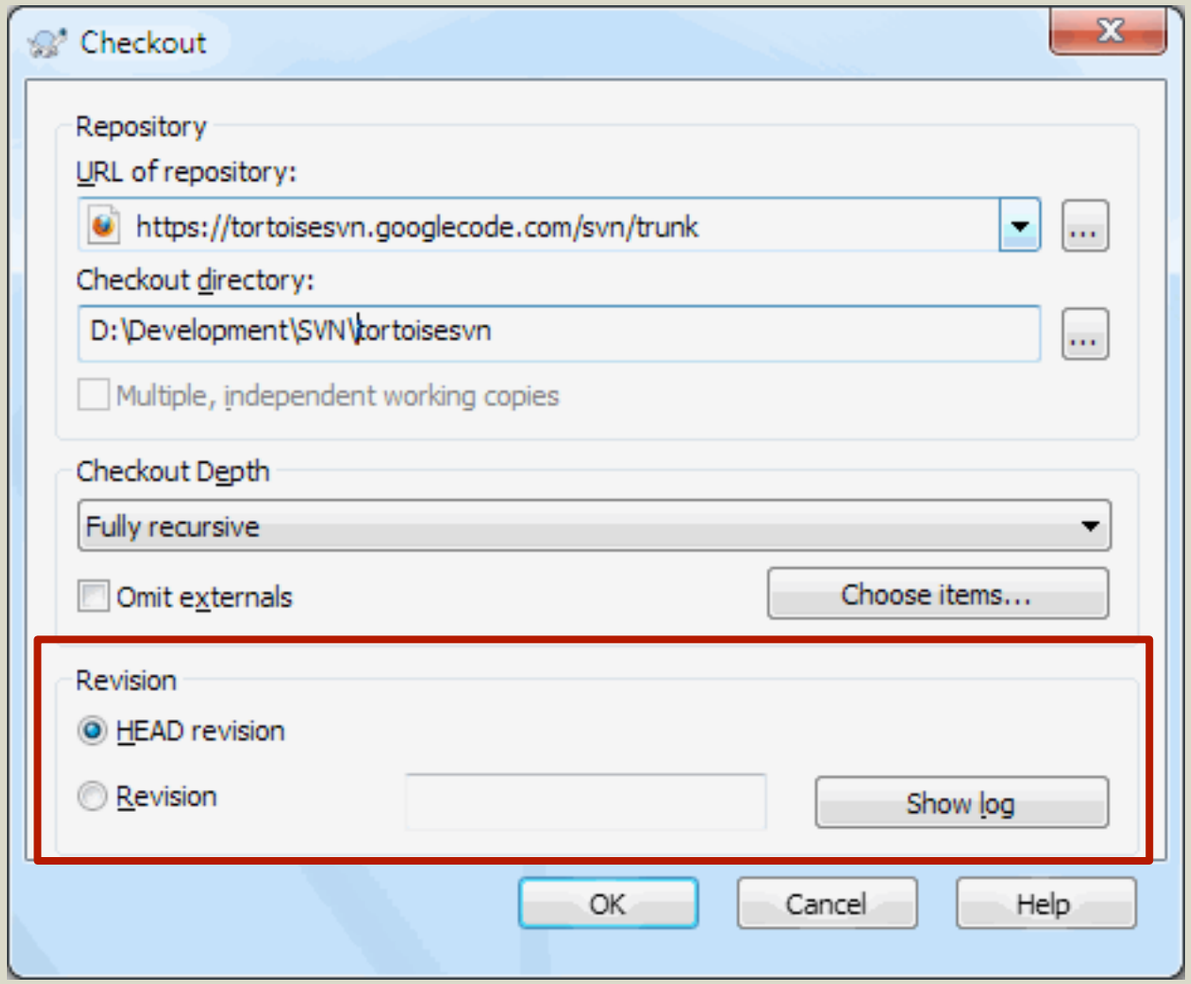
1) Checkout: obteniendo la copia de trabajo por primera vez

El *checkout* de un directorio del repositorio genera una copia de trabajo en el sistema de archivos local. Salvo que se especifique, esta copia contiene la última revisión.



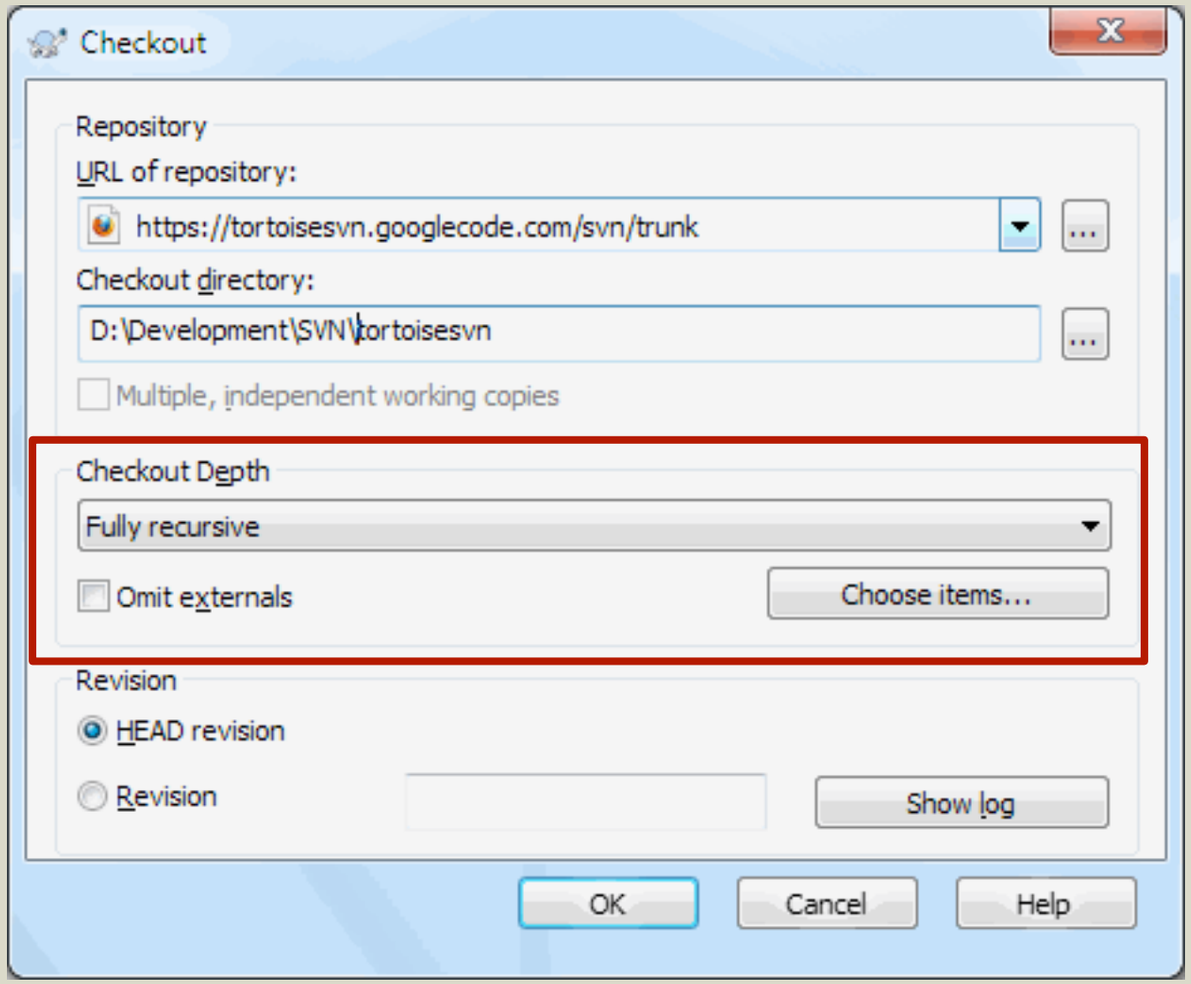
1) Checkout: obteniendo la copia de trabajo por primera vez

El *checkout* de un directorio del repositorio genera una copia de trabajo en el sistema de archivos local. Salvo que se especifique, esta copia contiene la última revisión.



1) Checkout: obteniendo la copia de trabajo por primera vez

El *checkout* de un directorio del repositorio genera una copia de trabajo en el sistema de archivos local. Salvo que se especifique, esta copia contiene la última revisión.



1) Checkout: obteniendo la copia de trabajo por primera vez

El *checkout* de un directorio del repositorio genera una copia de trabajo en el sistema de archivos local. Salvo que se especifique, esta copia contiene la última revisión.

```
$ svn checkout http://svn.ex.com/svn/repo/trunk
A    trunk/README
A    trunk/INSTALL
A    trunk/src/main.c
A    trunk/src/header.h
…
Checked out revision 8810.
$
```

1) Checkout: iconos de TortoiseSVN

El estado de cada archivo y directorio se refleja visualmente.

normal

eliminado

conflicto

modificado

ignorado

bloqueado

agregado

no versionado

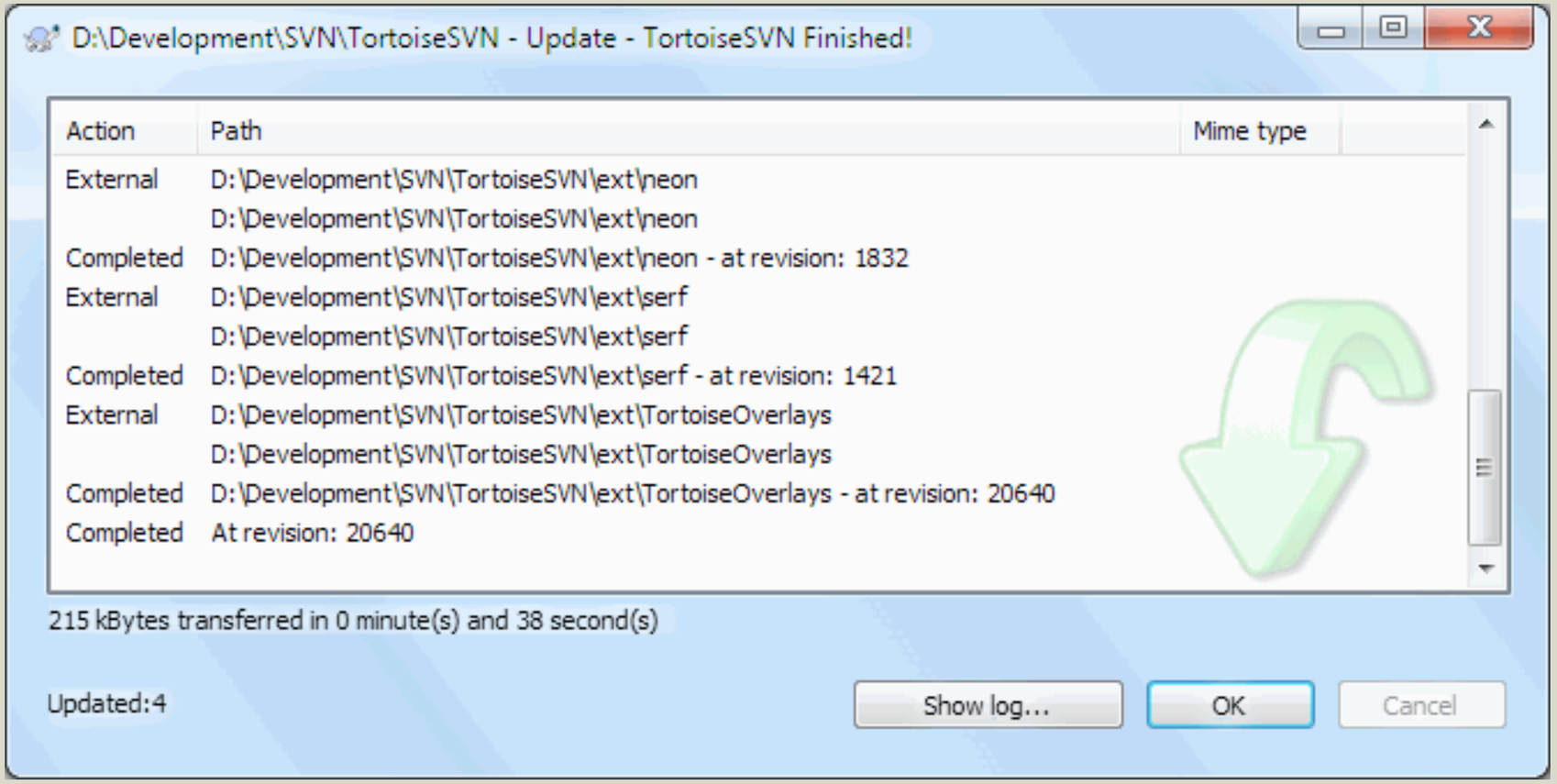
sólo lectura

Ciclo de trabajo

1) Update: actualizando la copia de trabajo

• Cuando se trabaja sobre un proyecto que puede ser modificado vía múltiples copias de trabajo, es recomendable realizar una actualización de la copia de trabajo para recibir los últimos cambios publicados por terceros.

• SVN no permite publicar cambios realizados sobre archivos desactualizados.



Ciclo de trabajo

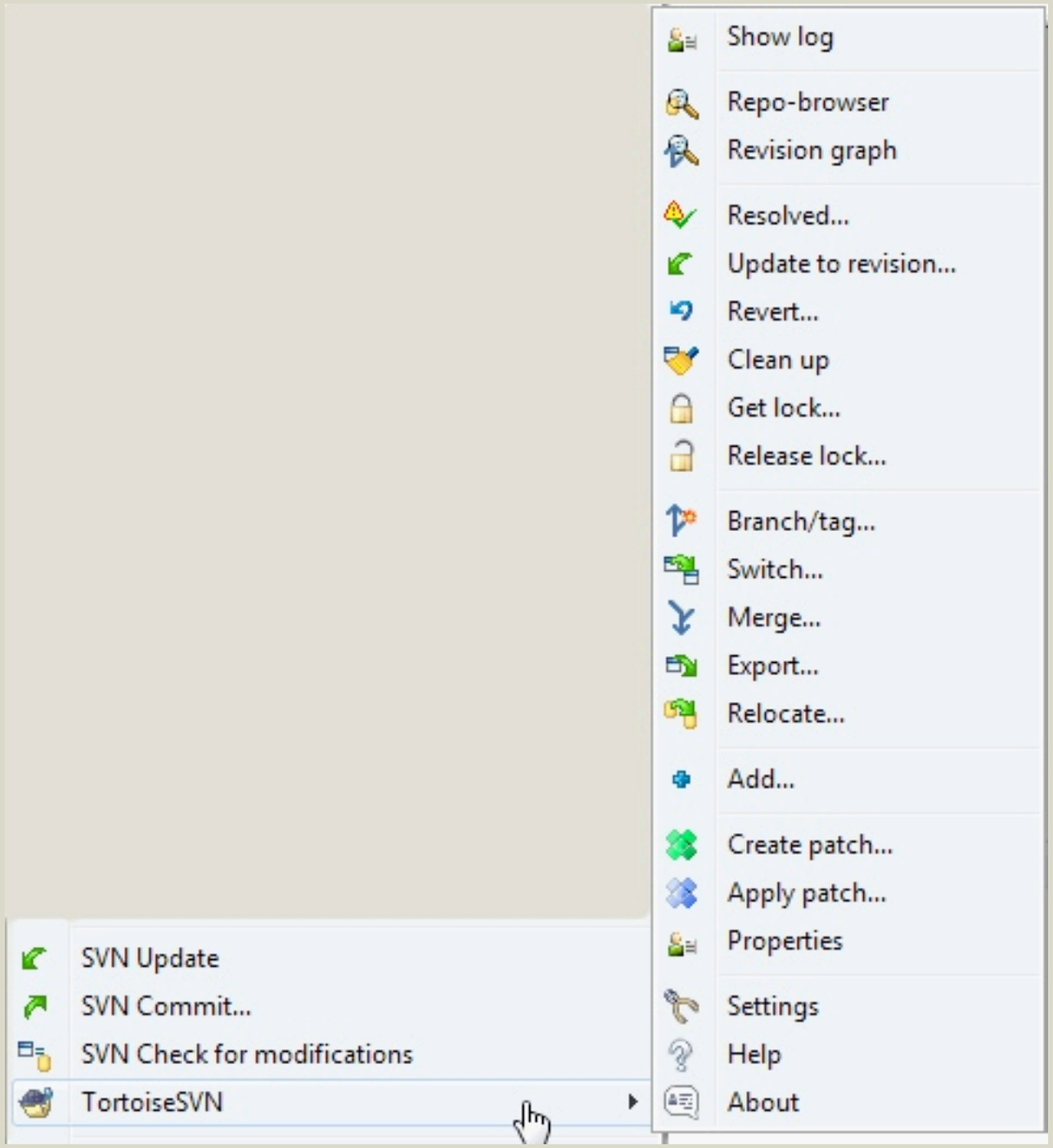
1) Update: actualizando la copia de trabajo

• Cuando se trabaja sobre un proyecto que puede ser modificado vía múltiples copias de trabajo, es recomendable realizar una actualización de la copia de trabajo para recibir los últimos cambios publicados por terceros.

• SVN no permite publicar cambios realizados sobre archivos desactualizados.

```
$ svn update
Updating '.':
U    foo.c
U    bar.c
Updated to revision 2.
$
```

2) Add: agregando componentes



- La opción de agregar programa la copia de trabajo para que agregue el componente al repositorio en la próxima publicación

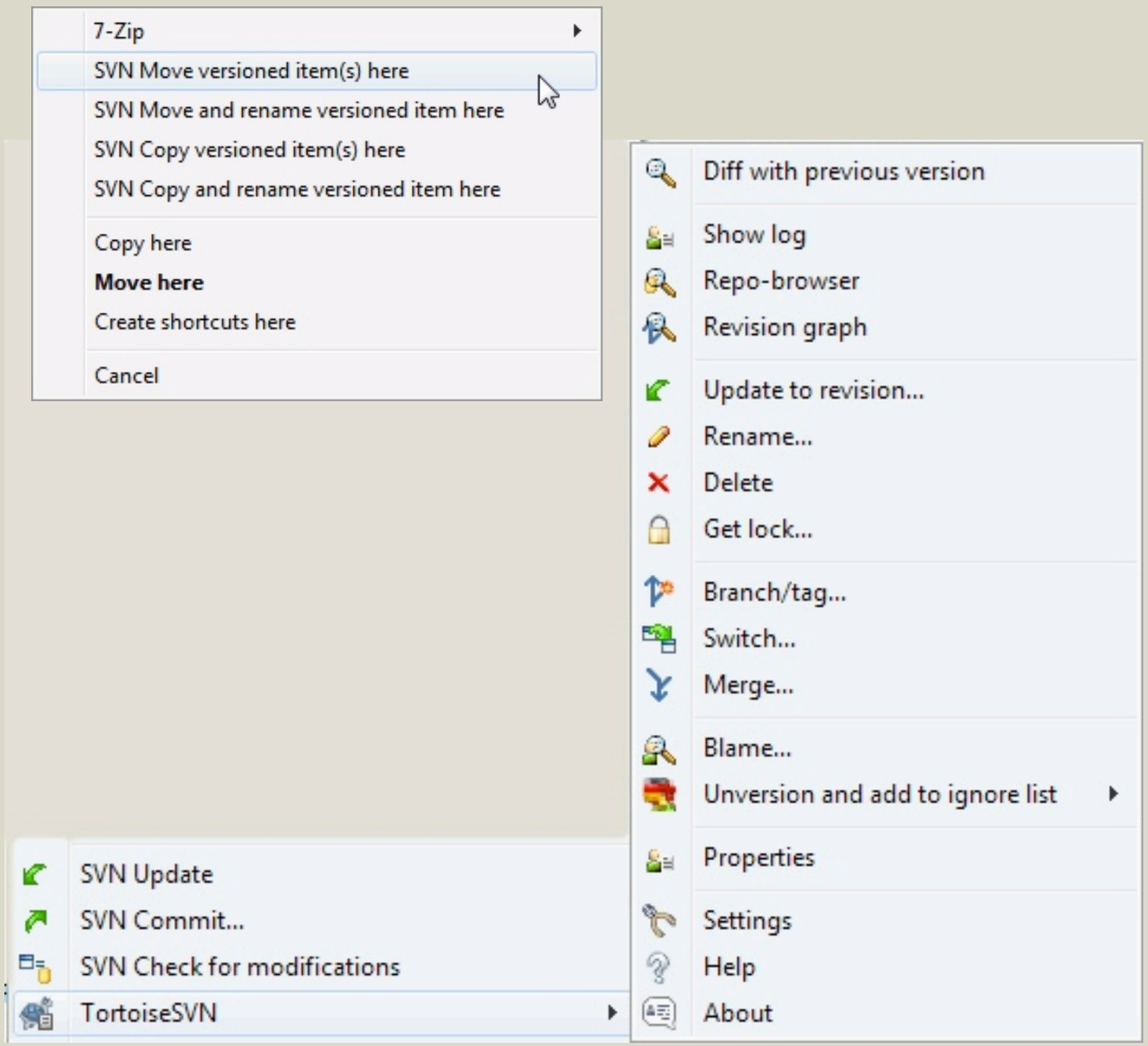
- Cuando se agrega un directorio, todo su contenido también es agregado.

2) Add: agregando componentes

```
$ svn add foo.c
A         foo.c

$ svn add testdir
A         testdir
A         testdir/a
A         testdir/b
A         testdir/c
A         testdir/d
```

- La opción de agregar programa la copia de trabajo para que agregue el componente al repositorio en la próxima publicación
- Cuando se agrega un directorio, todo su contenido también es agregado.

2) Move y Delete: moviendo y eliminando componentes



- Estas opciones programan la copia local para que mueva o elimine el componente.
- Si el componente marcado para eliminación es un archivo, entonces se lo elimina directamente.
- Siempre se debe manipular los componentes a través de los comandos de SVN!

2) Move y Delete: moviendo y eliminando componentes

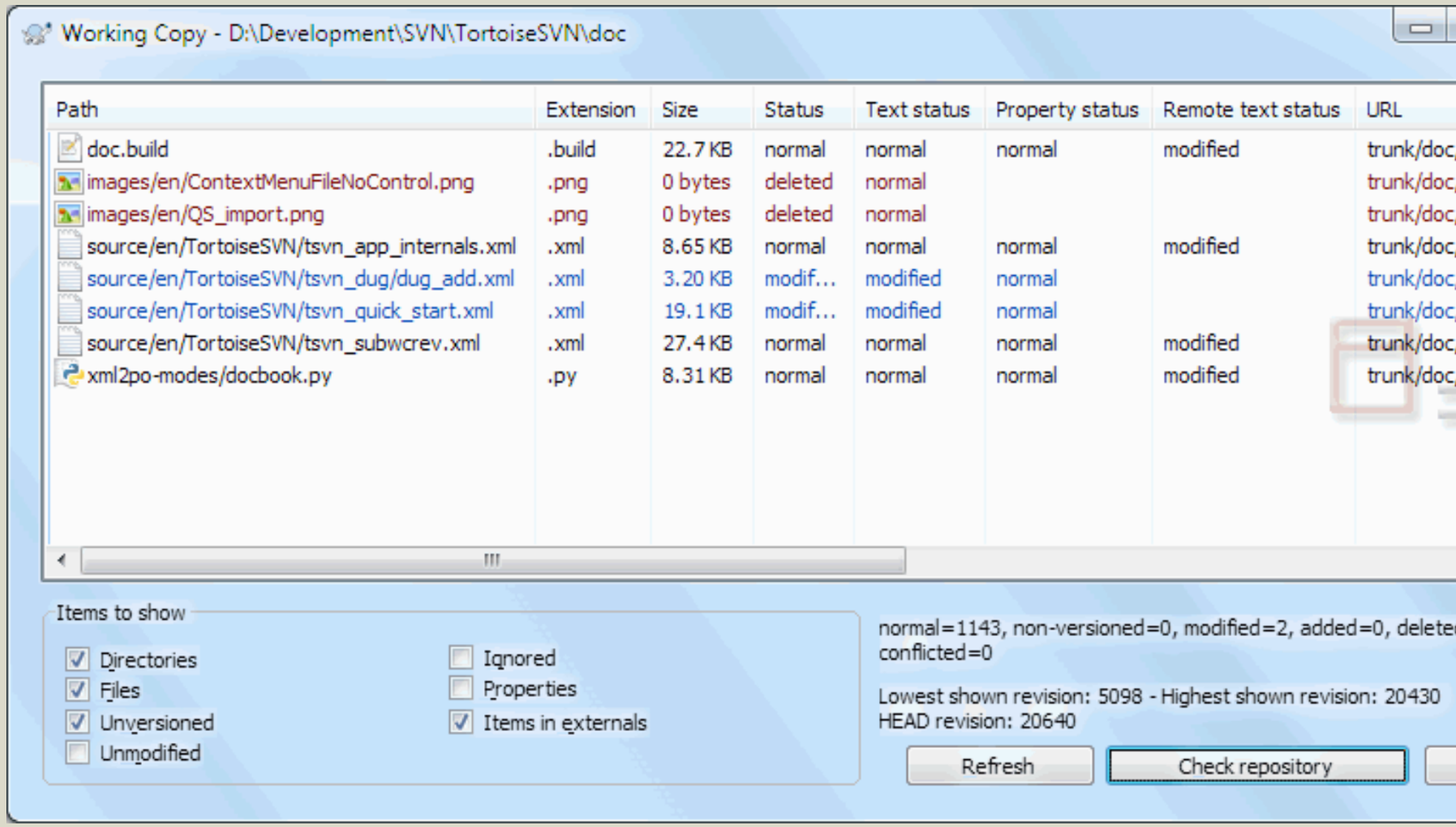
```
$ svn move foo.c bar.c
A         bar.c
D         foo.c

$ svn delete myfile
D         myfile

$ svn commit -m "Moving and deleting."
Adding          bar.c
Deleting        foo.c
Deleting        myfile
Transmitting file data .
Committed revision 14.
```

- Estas opciones programan la copia local para que mueva o elimine el componente.
- Si el componente marcado para eliminación es un archivo, entonces se lo elimina directamente.
- Siempre se debe manipular los componentes a través de los comandos de SVN!

3) Status: revisando los cambios

Una vez que se produjeron los cambios, y antes de publicarlos suele ser buena idea revisar los cambios, para:

- Registrar un mensaje apropiado en el log.
- Descubrir cambios que pasaron desapercibidos.
- Revisar críticamente los cambios introducidos.



3) Status: revisando los cambios

Una vez que se produjeron los cambios, y antes de publicarlos suele ser buena idea revisar los cambios, para:

- Registrar un mensaje apropiado en el log.
- Descubrir cambios que pasaron desapercibidos.
- Revisar críticamente los cambios introducidos.

```
$ svn status -v
M                44        23    sally     README
                 44        30    sally     INSTALL
M                44        20    harry     bar.c
                 44        18    ira       stuff
                 44        35    harry     stuff/trout.c
D                44        19    ira       stuff/fish.c
                 44        21    sally     stuff/things
A                 0         ?     ?        stuff/things/bl
                 44        36    harry     stuff/things/gl
```

3) Status: revisando los cambios

Una vez que se produjeron los cambios, y antes de publicarlos suele ser buena idea revisar los cambios, para:

- Registrar un mensaje apropiado en el log.
- Descubrir cambios que pasaron desapercibidos.
- Revisar críticamente los cambios introducidos.

Arreglando equivocaciones!

En algunas ocasiones los cambios a un componente resultan de una equivocación. Quizás el componente no debía ser modificado, o es mas fácil realizar los cambios apropiados comenzando desde el principio.

En cualquier circunstancia, podemos utilizar el comando **revert.**

```
$ svn status -v
M       44        23     sally
        44        30     sally
M       44        20     harry
        44        18     ira
        44        35     harry
D       44        19     ira
        44        21     sally
A       0         ?      ?
        44        36     harry
```

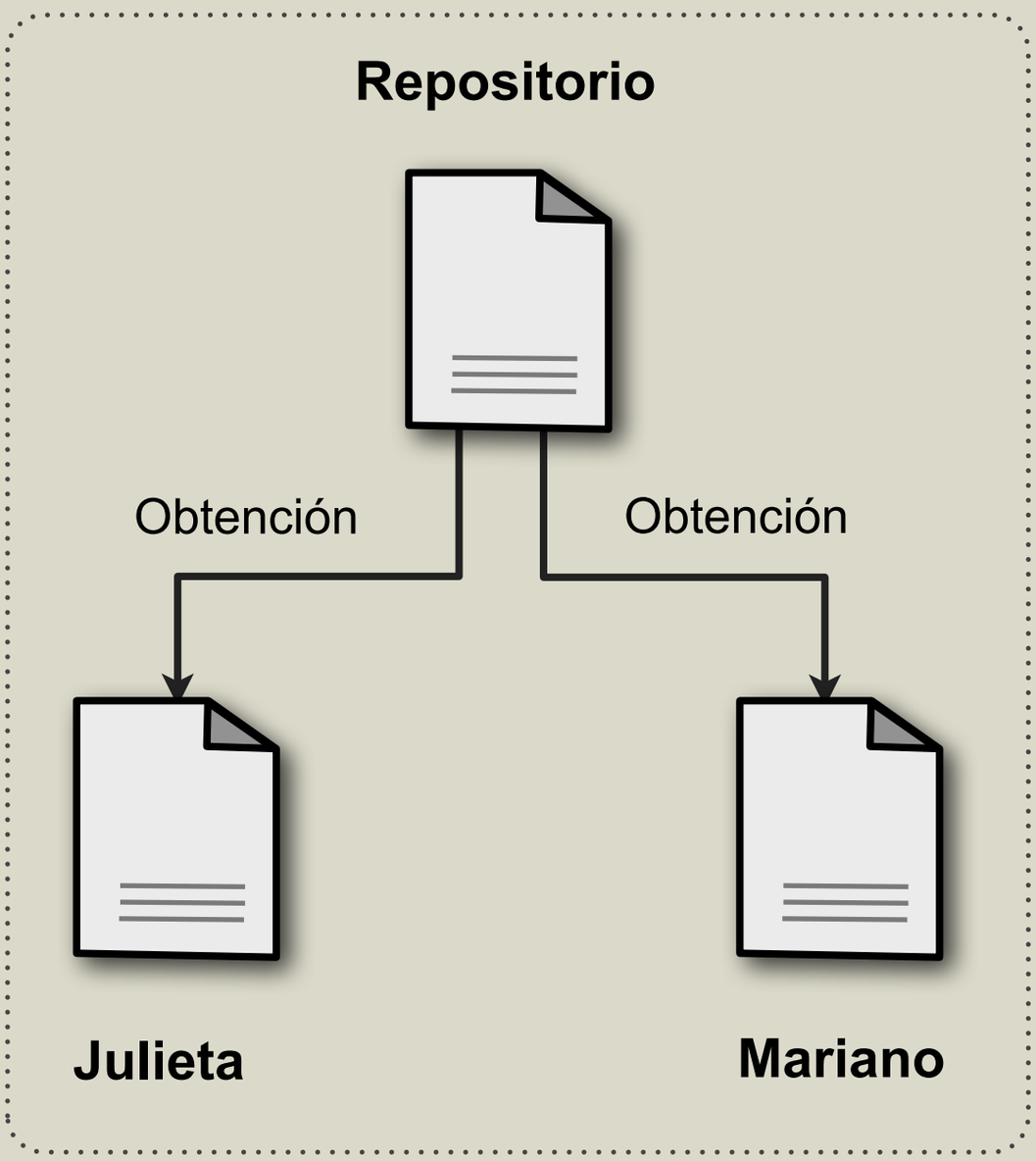
```
README
INSTALL
bar.c
stuff
stuff/trout.c
stuff/fish.c
stuff/things
stuff/things/bl
stuff/things/gl
```

4) Update: revisando la existencia de conflictos

Normalmente, antes de publicar las modificaciones es necesario actualizar la copia de trabajo. Esto puede dar lugar a dos situaciones:

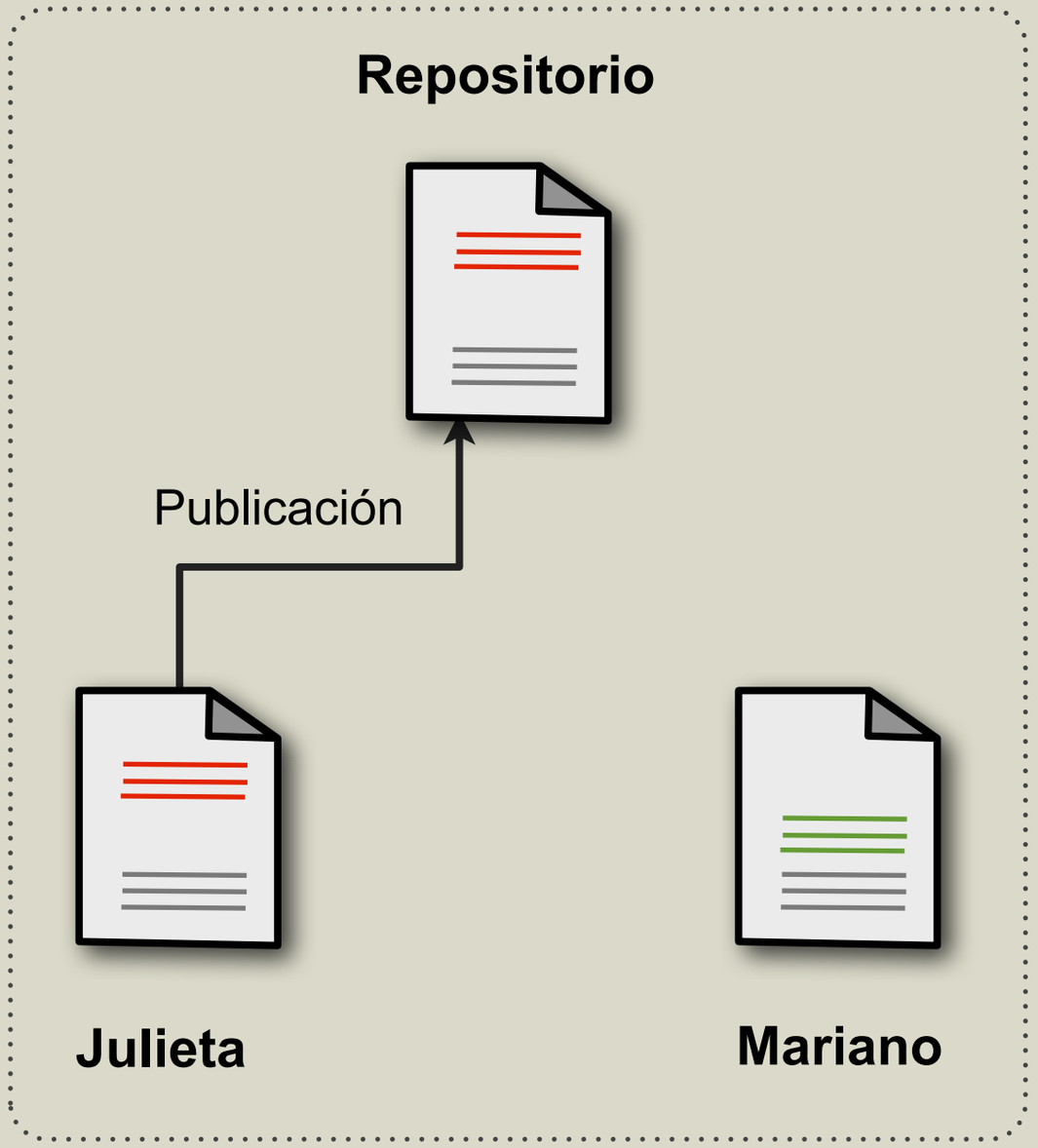
Situación sin conflictos

- Julieta y Mariano obtienen la información desde el repositorio.
- Ambos tienen la misma revisión.
- La información obtenida es su propia copia de trabajo.



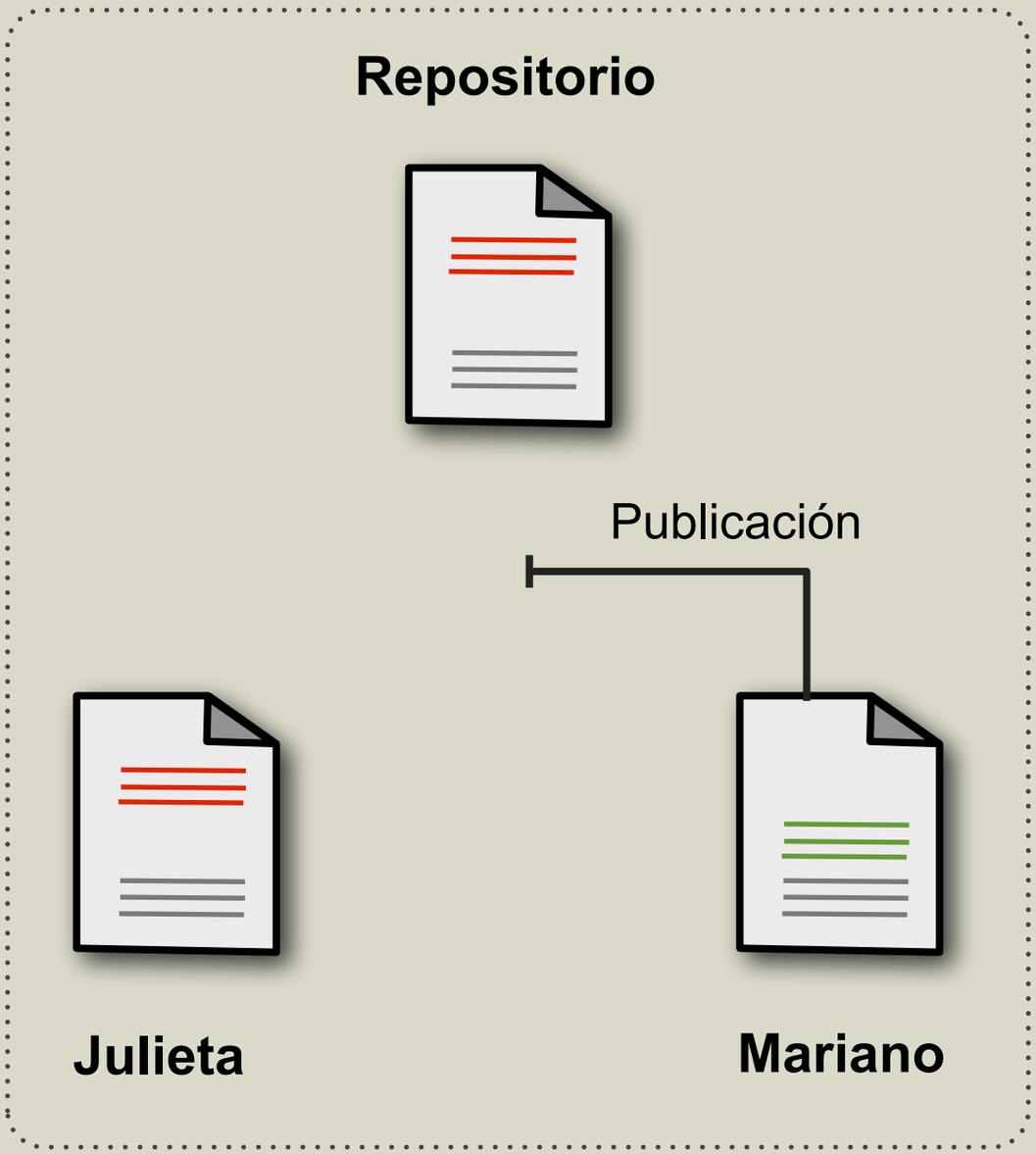
4) Update: revisando la existencia de conflictos

- Julieta y Mariano trabajan en diferentes lineas, y hacen diferentes modificaciones.
- Julieta publica sus cambios, creando una nueva revisión.



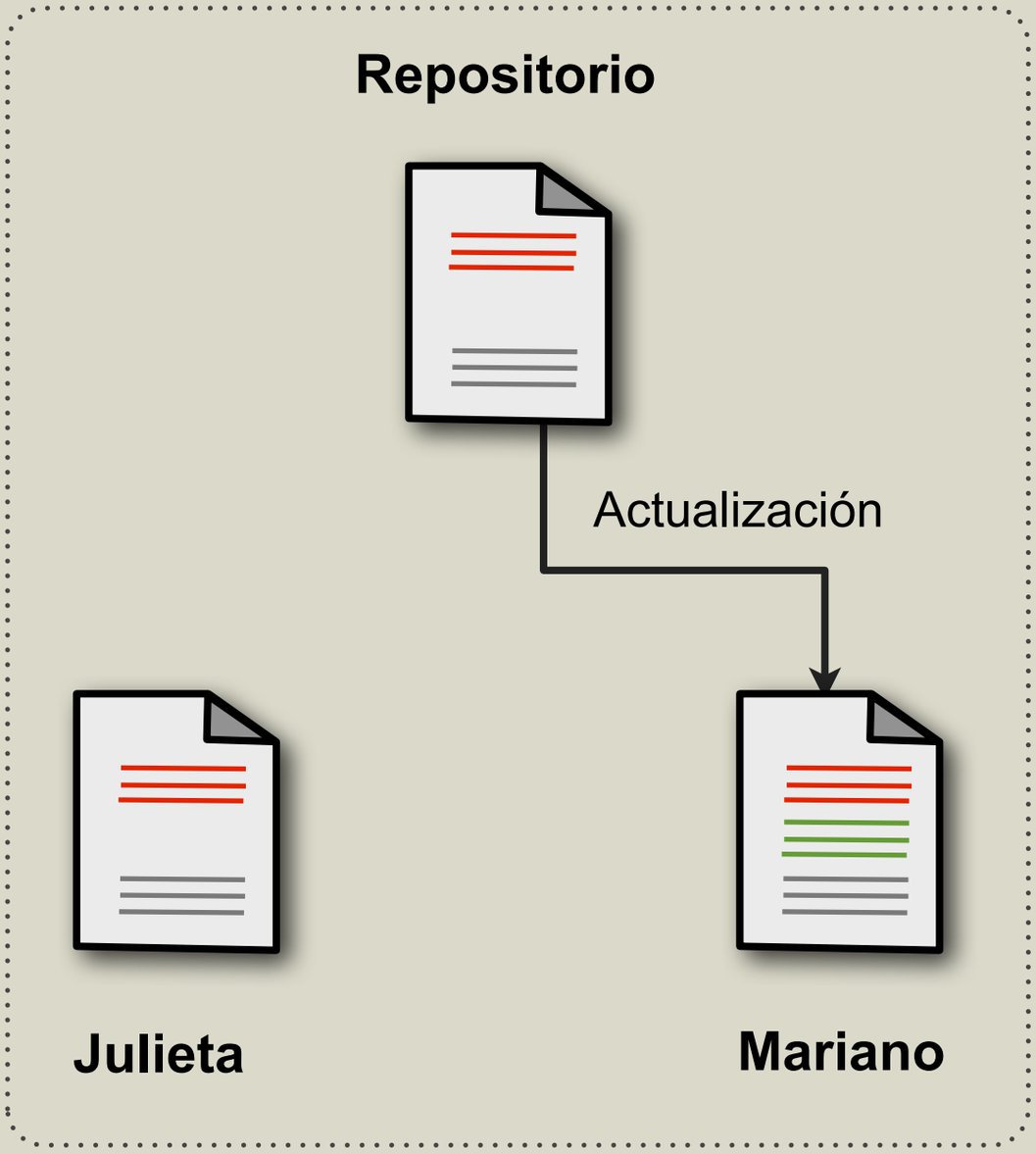
4) Update: revisando la existencia de conflictos

- Cuando Mariano intenta publicar sus propios cambios, obtiene un error.
- Su copia de trabajo está desactualizada: no posee los cambios introducidos por Julieta.



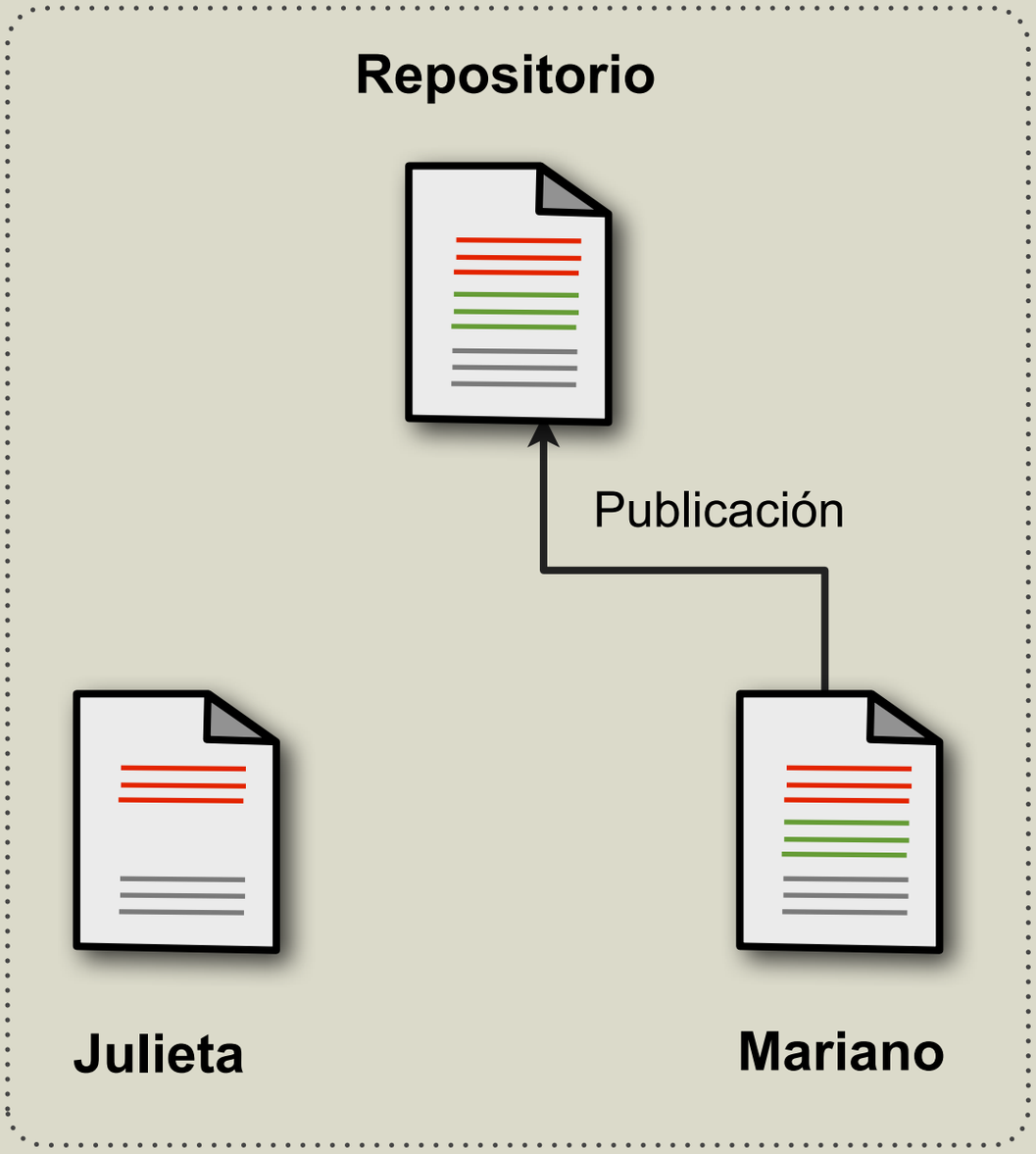
4) Update: revisando la existencia de conflictos

- Mariano debe actualizar su copia de trabajo para poder continuar.
- SVN integra los cambios de Julieta automáticamente.



4) Update: revisando la existencia de conflictos

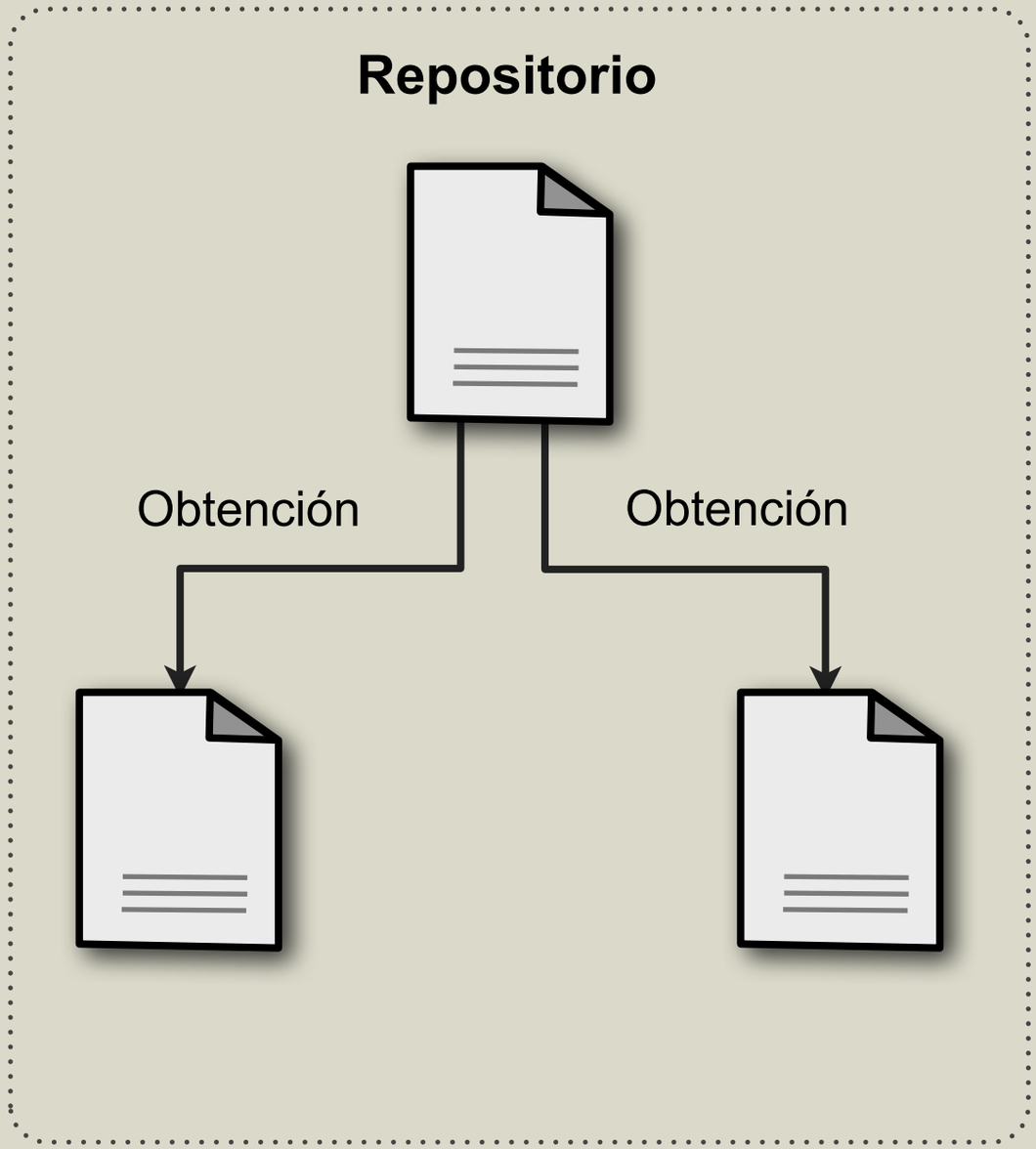
- Ahora Mariano puede publicar sus cambios integrados con los cambios de Julieta.
- Se genera una nueva revisión.



4) Update: revisando la existencia de conflictos

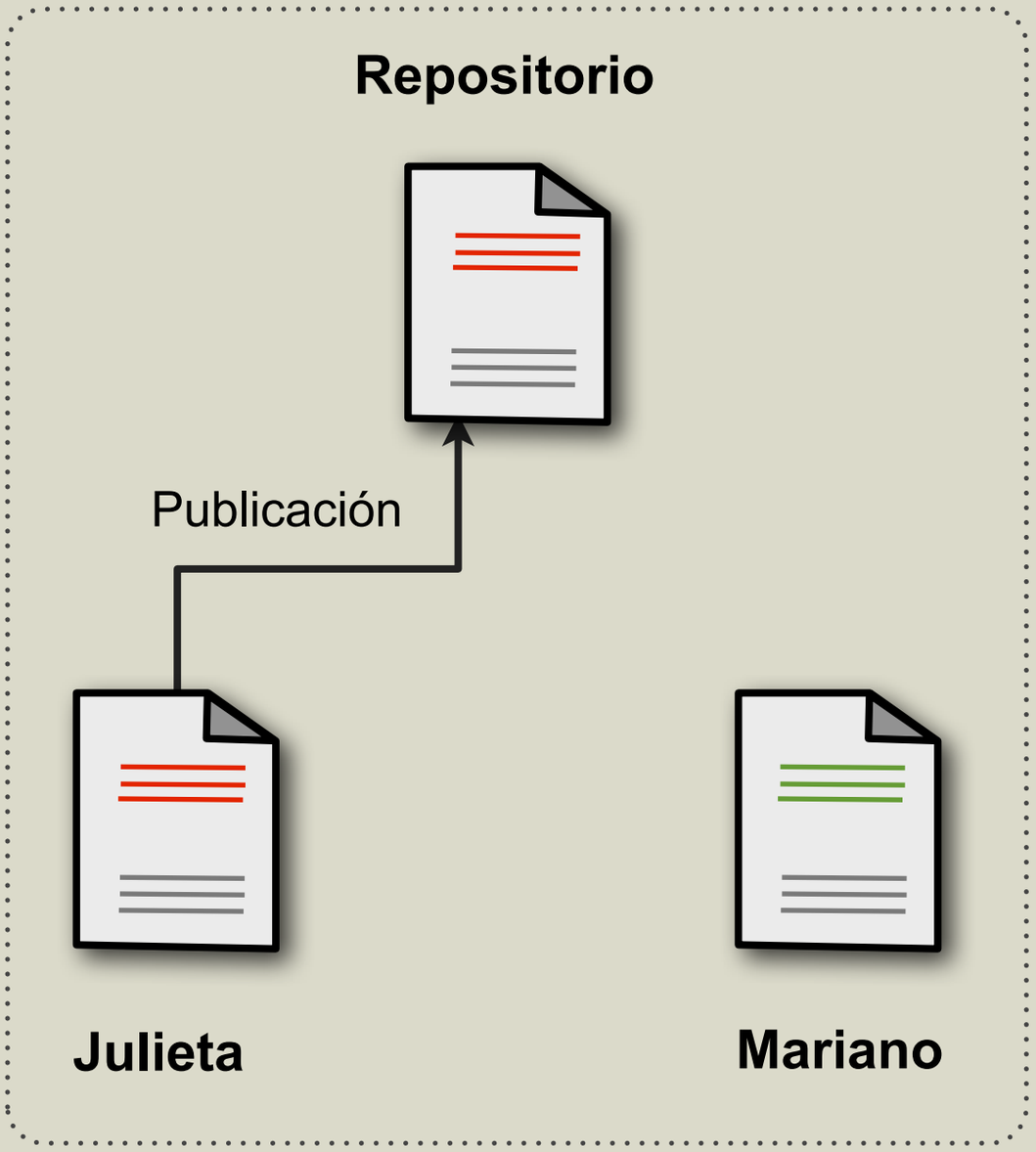
Situación con conflictos

- Julieta y Mariano obtienen la información desde el repositorio.
- Ambos tienen la misma revisión.
- La información obtenida es su propia copia de trabajo.



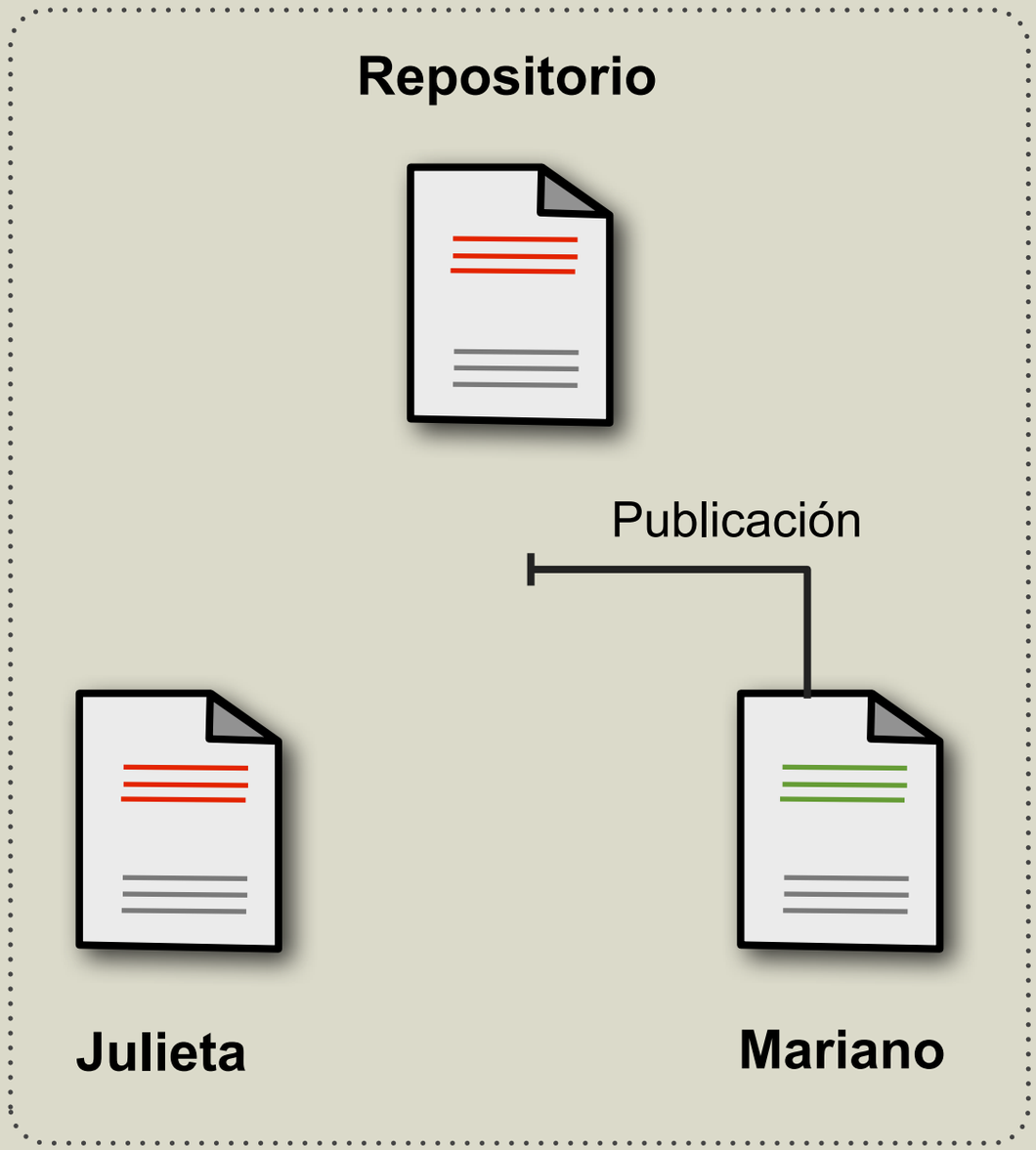
4) Update: revisando la existencia de conflictos

- Julieta y Mariano trabajan en la misma porción del archivo.
- Julieta publica sus cambios, creando una nueva revisión.



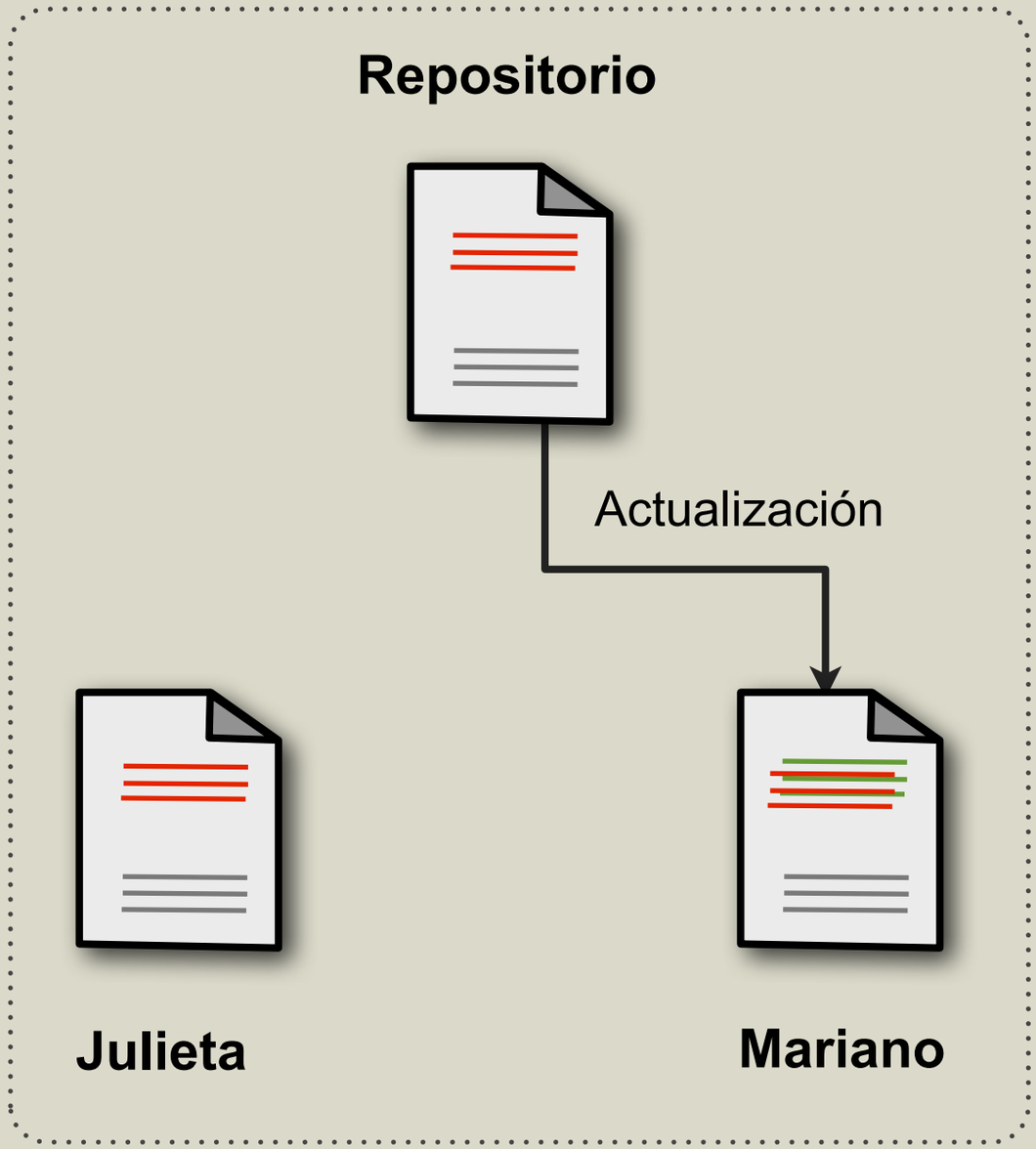
4) Update: revisando la existencia de conflictos

- Cuando Mariano intenta publicar sus propios cambios, obtiene un error.
- Su copia de trabajo está desactualizada: no posee los cambios introducidos por Julieta.

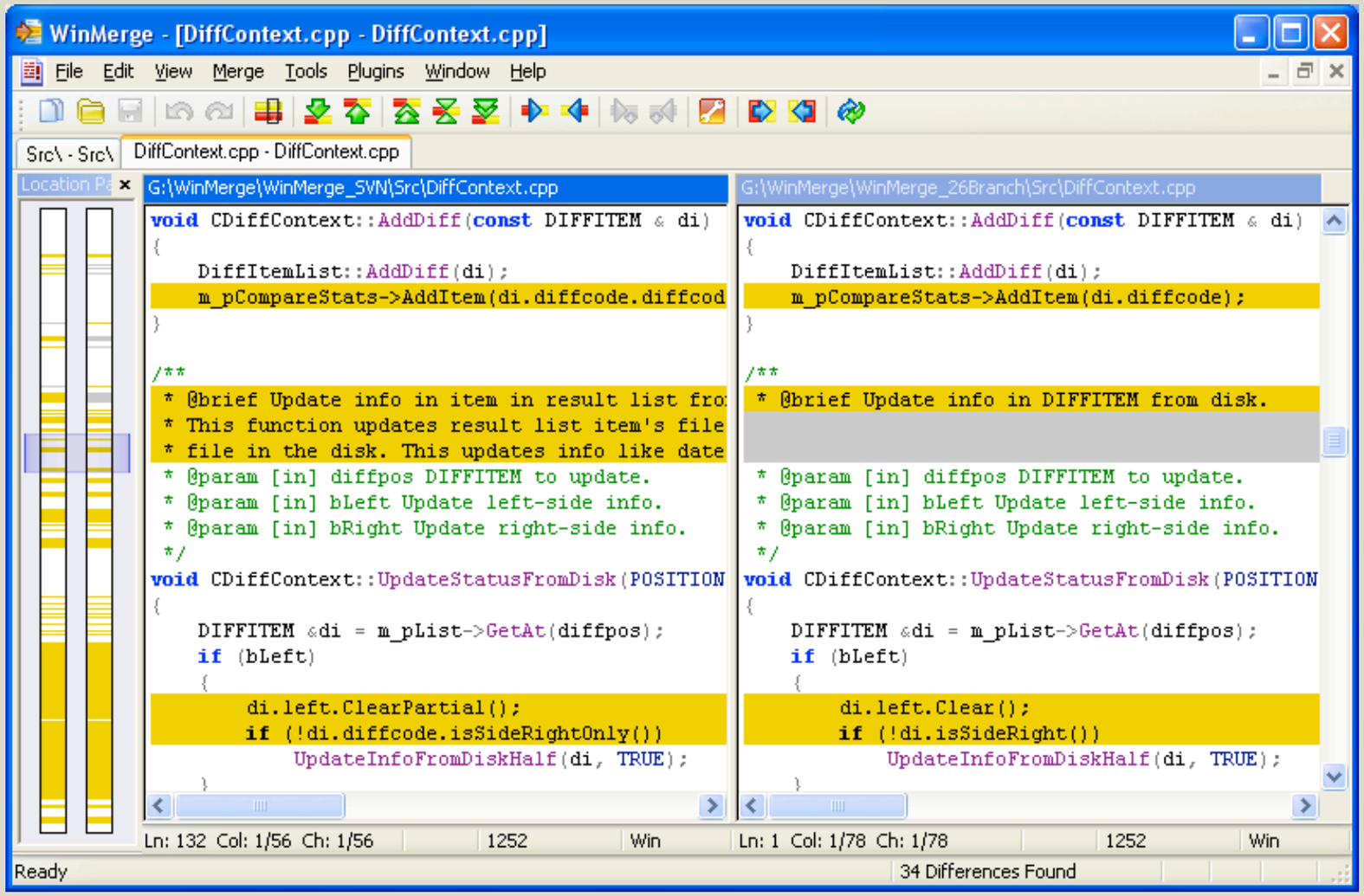


4) Update: revisando la existencia de conflictos

- Mariano debe actualizar su copia de trabajo para poder continuar.
- **Pero los cambios son inconsistentes!**



5) Diff y Resolve: arreglando conflictos



- Los conflictos pueden editarse observando las diferencias línea-a-línea entre las revisiones.
- La resolución de los conflictos queda a cargo del usuario, y puede forzarse.
- Aún cuando no existan conflictos **sintácticos**, pueden existir conflictos **semánticos**.

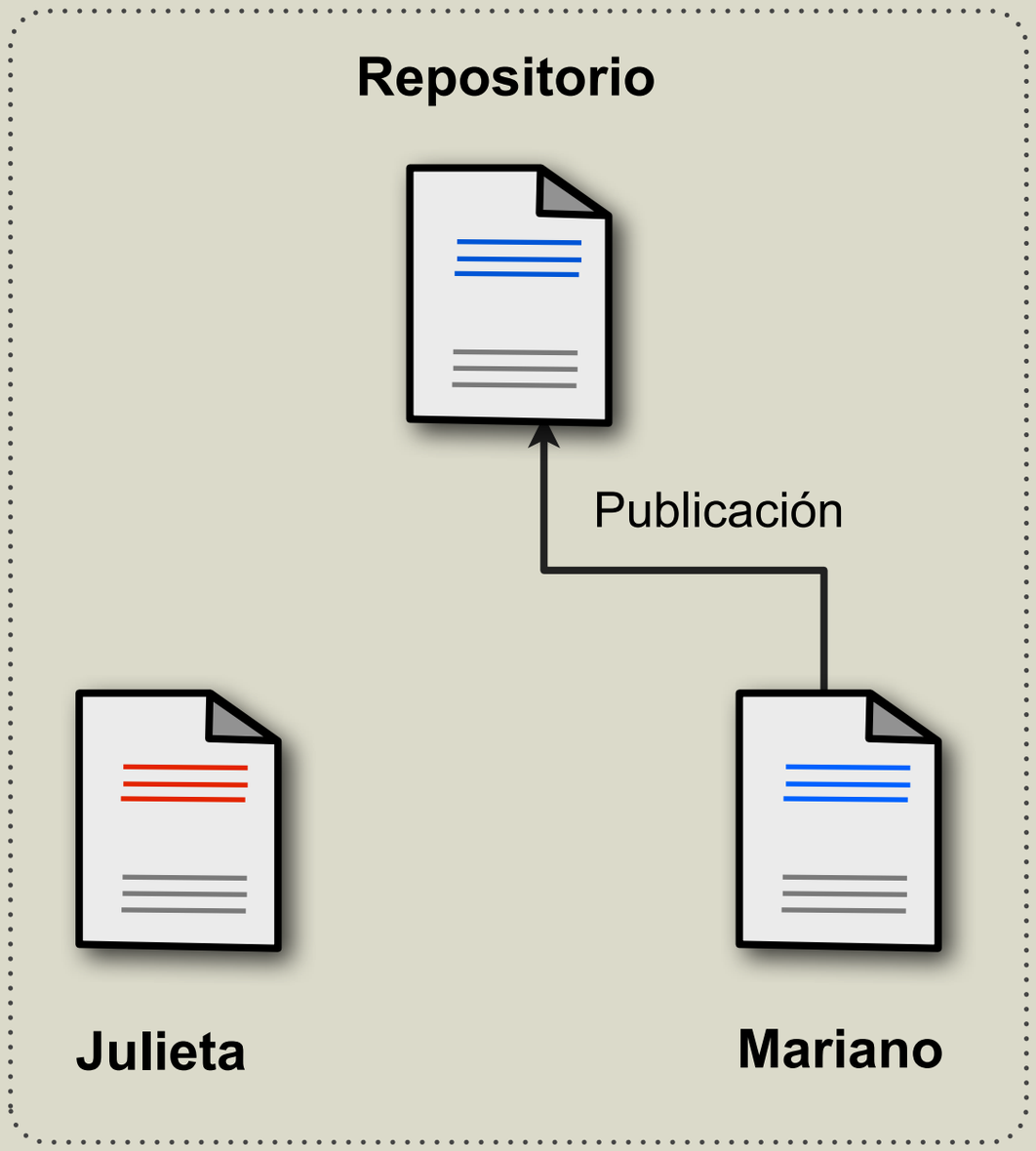
5) Diff y Resolve: arreglando conflictos

```
$ svn diff
Index: bar.c
===================================================================
--- bar.c  (revision 3)
+++ bar.c  (working copy)
@@ -1,7 +1,12 @@
+#include <sys/types.h>
+#include <sys/stat.h>
+#include <unistd.h>
+
+#include <stdio.h>

 int main(void) {
-  printf("Sixty-four slices of American Cheese...\n");
+  printf("Sixty-five slices of American Cheese...\n");
  return 0;
 }

Index: README
===================================================================
--- README  (revision 3)
+++ README  (working copy)
@@ -193,3 +193,4 @@
+Note to self:  pick up laundry.
...
```

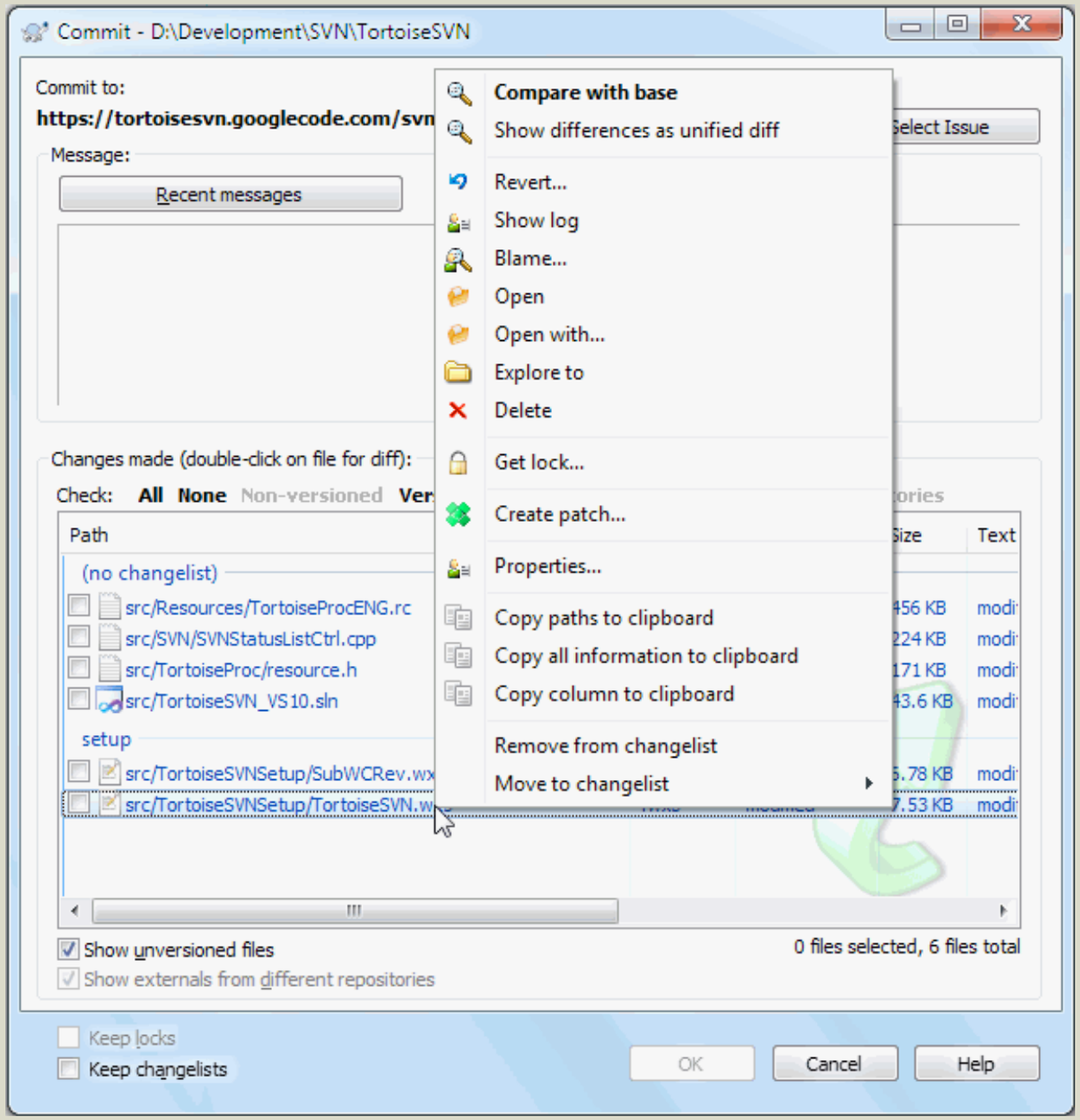
- Los conflictos pueden editarse observando las diferencias línea-a-línea entre las revisiones.
- La resolución de los conflictos queda a cargo del usuario, y puede forzarse.
- Aún cuando no existan conflictos **sintácticos**, pueden existir conflictos **semánticos**.

5) Diff y Resolve: arreglando conflictos

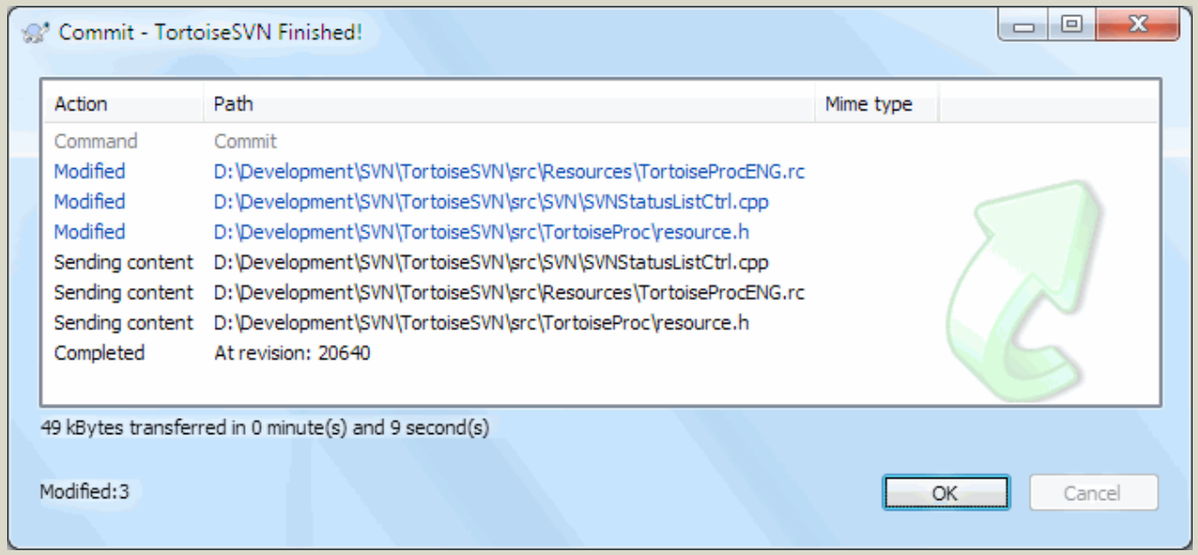
- Una vez resueltos los conflictos, y marcados como tales, Mariano puede publicar una versión combinada.
- Se genera una nueva revisión.



6) Commit: publicando los cambios



Cuando se han resuelto todos los conflictos, se publican los cambios en el repositorio, generándose una nueva revisión.



6) Commit: publicando los cambios

Cuando se han resuelto todos los conflictos, se publican los cambios en el repositorio, generándose una nueva revisión.

```
$ svn commit -m "Corrected number of cheese slices."
Sending        sandwich.txt
Transmitting file data .
Committed revision 3.
```

Sumario

Introducción

Conceptos básicos | Arquitectura de *Subversion*

Ciclo de trabajo básico

Obteniendo contenido | Haciendo cambios | Revisando cambios

| Revisando conflictos | Resolviendo conflictos | Publicando cambios

Temas avanzados

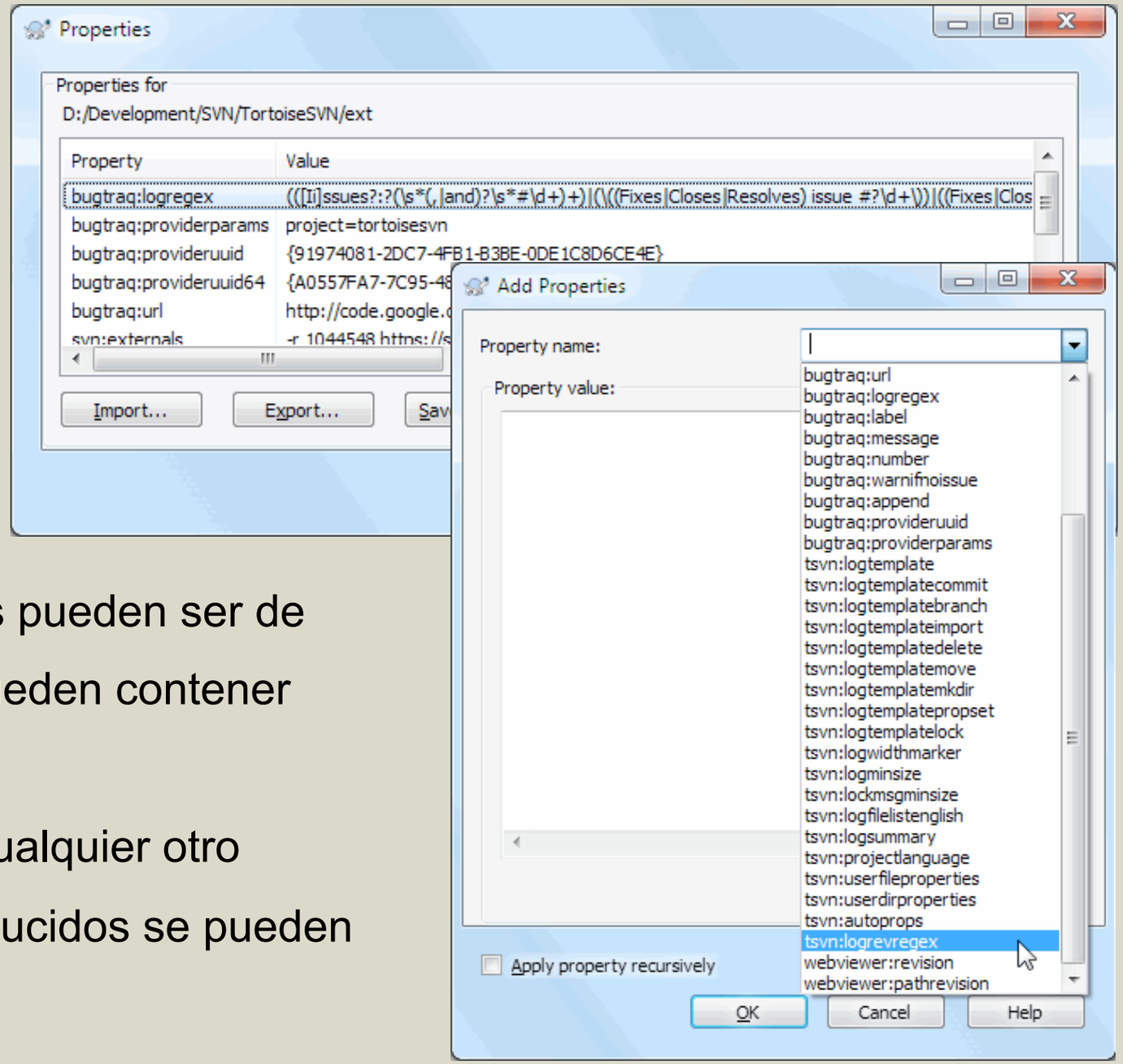
Propiedades | Modo bloqueante

Bifurcación y combinación

¿Por y para qué? | Bifurcaciones | Sincronización | Reintegración

Propiedades

- *Subversion* provee interfaces para agregar, modificar y eliminar metadatos (versionados) en cada archivo y directorio.
- Estos metadatos son llamados propiedades: cada propiedad tiene un nombre y un valor asociado.
- Los nombres y valores de las propiedades pueden ser de cualquier tipo, con la restricción que sólo pueden contener caracteres ASCII.
- Las propiedades son versionadas como cualquier otro contenido de los archivos; los cambios producidos se pueden publicar y revertir.



Propiedades

• *Subversion* provee interfaces para agregar, modificar y eliminar metadatos (versionados) en cada archivo y directorio.

• Estos metadatos son llamados propiedades: cada propiedad tiene un nombre y un valor asociado.

```
$ svn propset copyright '(c) 2006 Red-Bean' calc/button.c
property 'copyright' set on 'calc/button.c'

$ svn propset license -F /path/to/LICENSE calc/button.c
property 'license' set on 'calc/button.c'

$ svn proplist calc/button.c
Properties on 'calc/button.c':
  copyright
  license

$ svn propget copyright calc/button.c
(c) 2006 Red-Bean
```

• Los nombres y valores de las propiedades pueden ser de cualquier tipo, con la restricción que sólo pueden contener caracteres ASCII.

• Las propiedades son versionadas como cualquier otro contenido de los archivos; los cambios producidos se pueden publicar y revertir.

Propiedades

svn:ignore:

- Indica qué archivos y/o directorios no deben versionarse: archivos de *backup* generados automáticamente por los editores, resultados (intermedios) de compilación, etc.
- Se aplica a directorios.
- El valor es un *pattern*. Por ejemplo: `*.exe`, `*.dll`

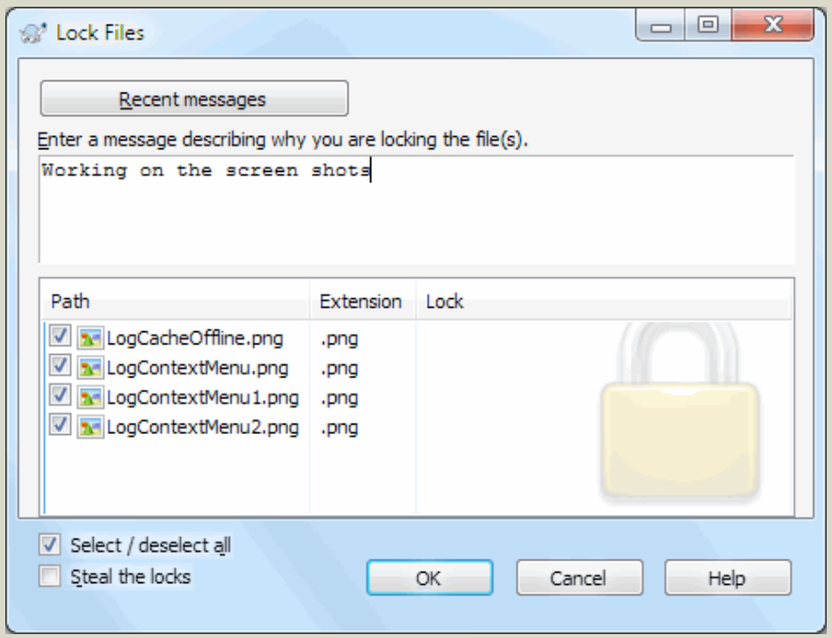
svn:externals:

- Indica que el componente pertenece a otro repositorio.
- Permite crear copias de trabajo de diferentes fuentes.
- Se aplica tanto a directorios como a archivos individuales.
- La administración de los componentes externos es independiente.

Modo bloqueante

- El modelo copiar-modificar-combinar funciona muy bien con archivos de texto
- Pero no es adecuado para manejar archivos binarios, ya que es difícil *localizar* los cambios y compararlos
- El bloqueo permite *serializar* modificaciones que no se pueden paralelizar.

Bloquear → **Editar** → **Desbloquear**



Modo bloqueante

- El modelo copiar-modificar-combinar funciona muy bien con archivos de texto
- Pero no es adecuado para manejar archivos binarios, ya que es difícil *localizar* los cambios y compararlos
- El bloqueo permite *serializar* modificaciones que no se pueden paralelizar.

Bloquear → **Editar** → **Desbloquear**

```
$ svn lock banana.jpg -m "Editing file for tomorrow's
release."
'banana.jpg' locked by user 'harry'.

$ svn status
     K  banana.jpg
$
```

Modo bloqueante

- El modelo copiar-modificar-combinar funciona muy bien con archivos de texto
- Pero no es adecuado para manejar archivos binarios, ya que es difícil *localizar* los cambios y compararlos
- El bloqueo permite *serializar* modificaciones que no se pueden paralelizar.

- SVN evita que otros usuarios modifiquen el archivo mientras está bloqueado.
- La publicación de cambios, libera el bloqueo.
- Los bloqueos pueden romperse y robarse.
- La necesidad de bloqueo puede recordarse con la propiedad `svn:needs-lock`.

Sumario

Introducción

Conceptos básicos | Arquitectura de *Subversion*

Ciclo de trabajo básico

Obteniendo contenido | Haciendo cambios | Revisando cambios

| Revisando conflictos | Resolviendo conflictos | Publicando cambios

Temas avanzados

Propiedades | Modo bloqueante

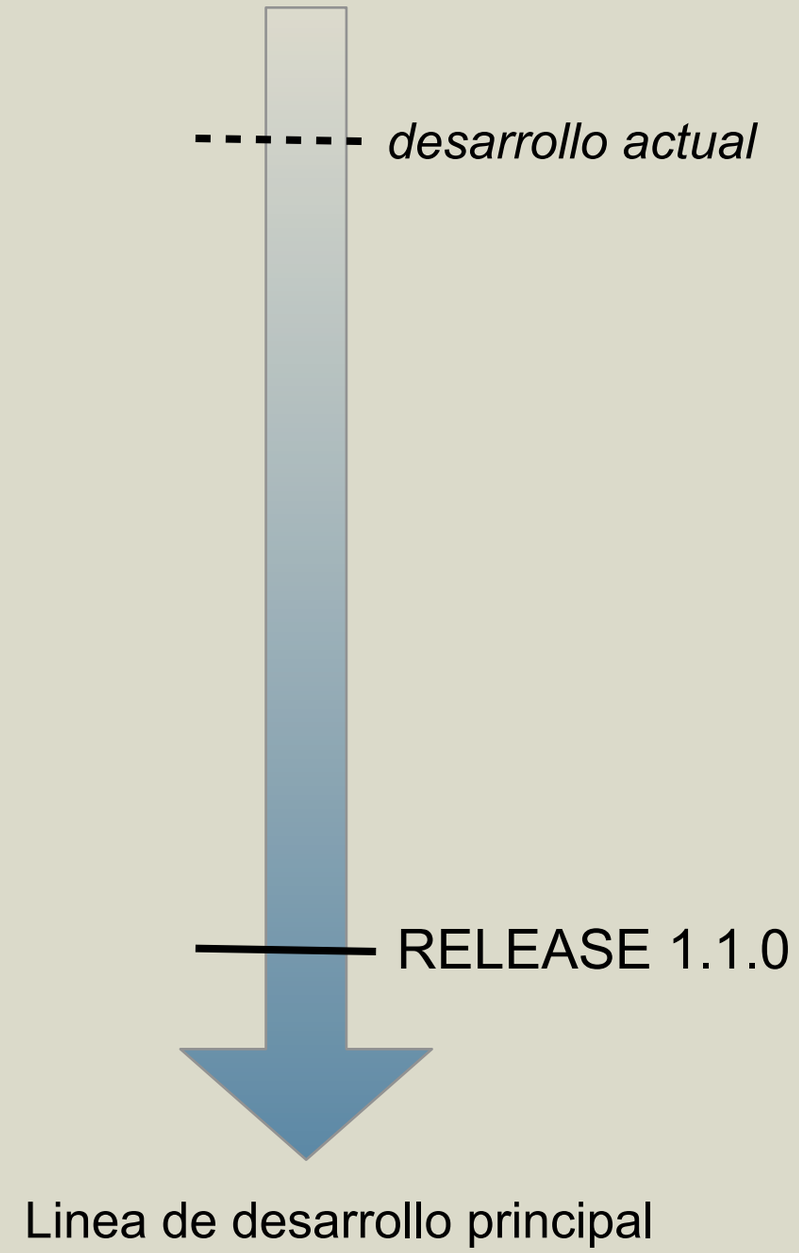
Bifurcación y combinación

¿Por y para qué? | Bifurcaciones | Sincronización | Reintegración

¿Por y para qué?

Supongamos esta situación:

- Se implementó un conjunto suficiente de funcionalidades de un componente de software para ser enviado a producción.
- Se pretende que el equipo de desarrollo siga trabajando sobre el mismo componente, para incluir nuevas funcionalidades y liberarlas en el futuro.

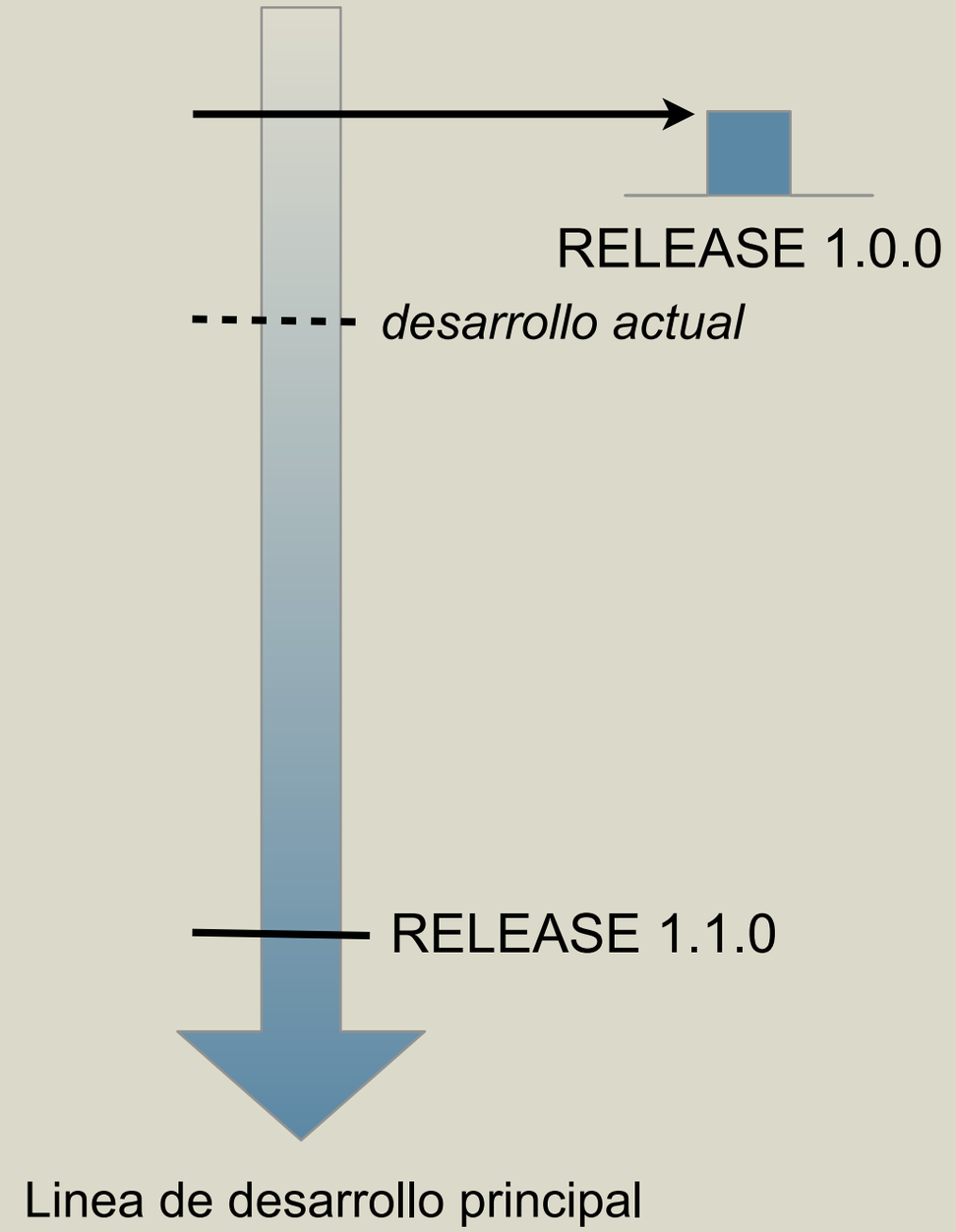


¿Por y para qué?

Supongamos esta situación:

- Se implementó un conjunto suficiente de funcionalidades de un componente de software para ser enviado a producción.
- Se pretende que el equipo de desarrollo siga trabajando sobre el mismo componente, para incluir nuevas funcionalidades y liberarlas en el futuro.

Se genera una *tag: una* instantánea del estado actual (una revisión particular) para referencia.



Bifurcaciones y combinaciones

¿Por y para qué?

Supongamos esta otra situación:

- Se implementó un componente de software y fue liberado como la versión RELEASE 1.0.0
- El equipo de desarrollo sigue trabajando, incluyendo nuevas funcionalidades para liberar en el futuro la versión RELEASE 1.1.0
- Un usuario encuentra un defecto!

En **necesario reparar el defecto**, pero:

- En el desarrollo actual hay nuevas funcionalidades que no se quieren entregar antes de tiempo

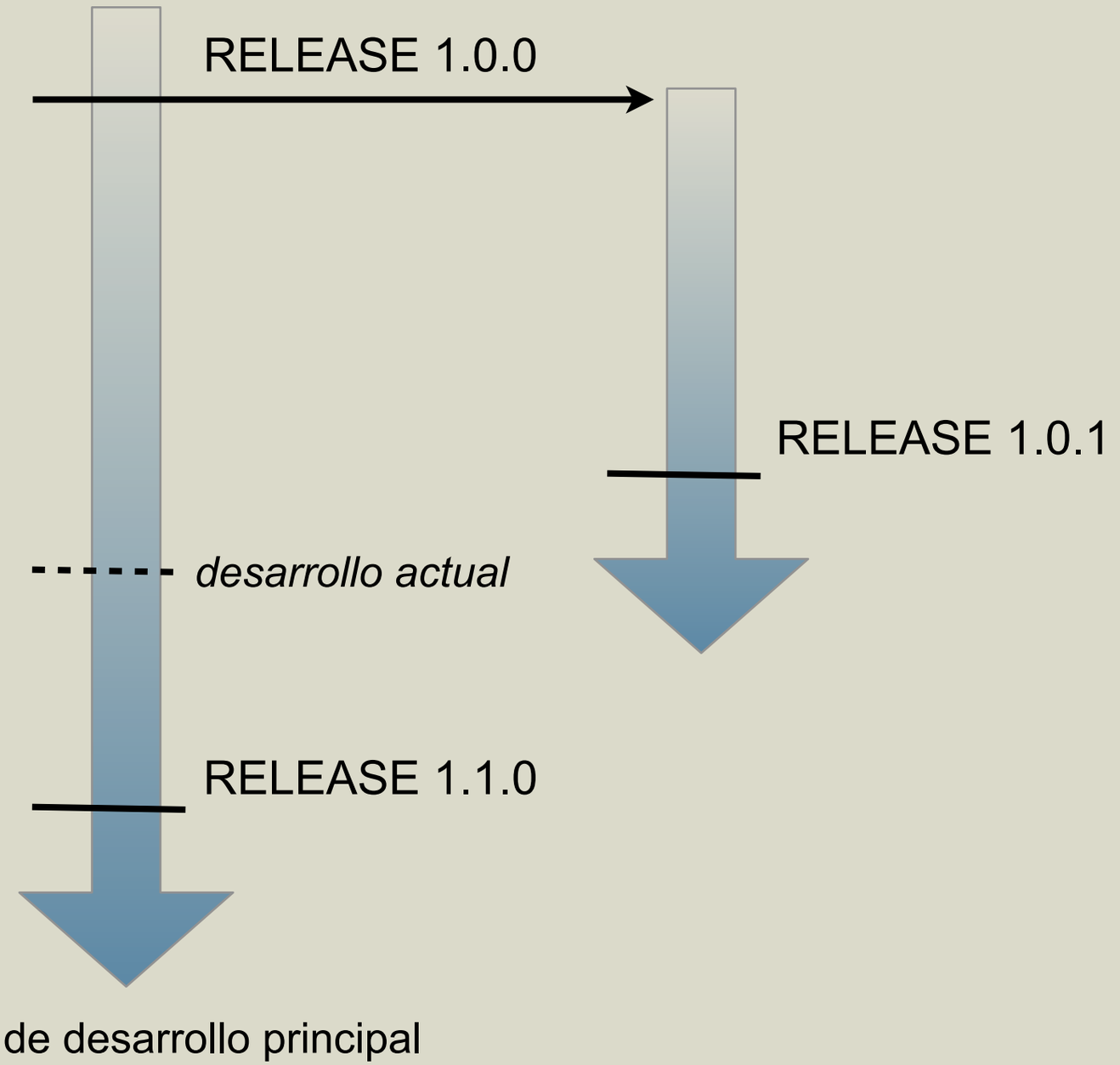
¿Por y para qué?

¿Como resolver esta situación?

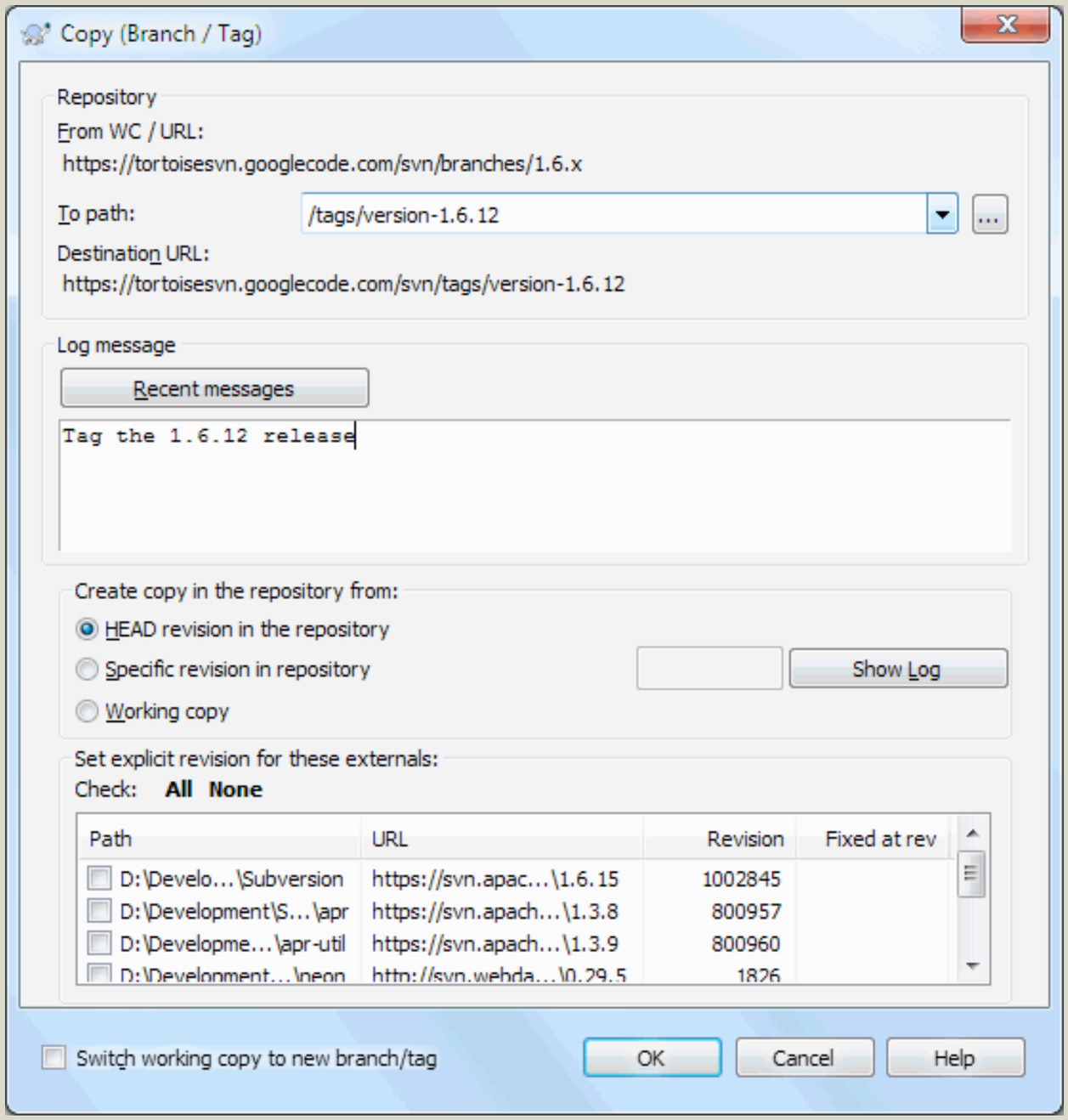
- Se recupera la revisión RELEASE 1.0.0
- Se crea un *branch* sobre el cual trabajar para reparar el defecto
- Después de reparado el defecto, se libera una versión corregida, como RELEASE 1.0.1

Como consecuencia:

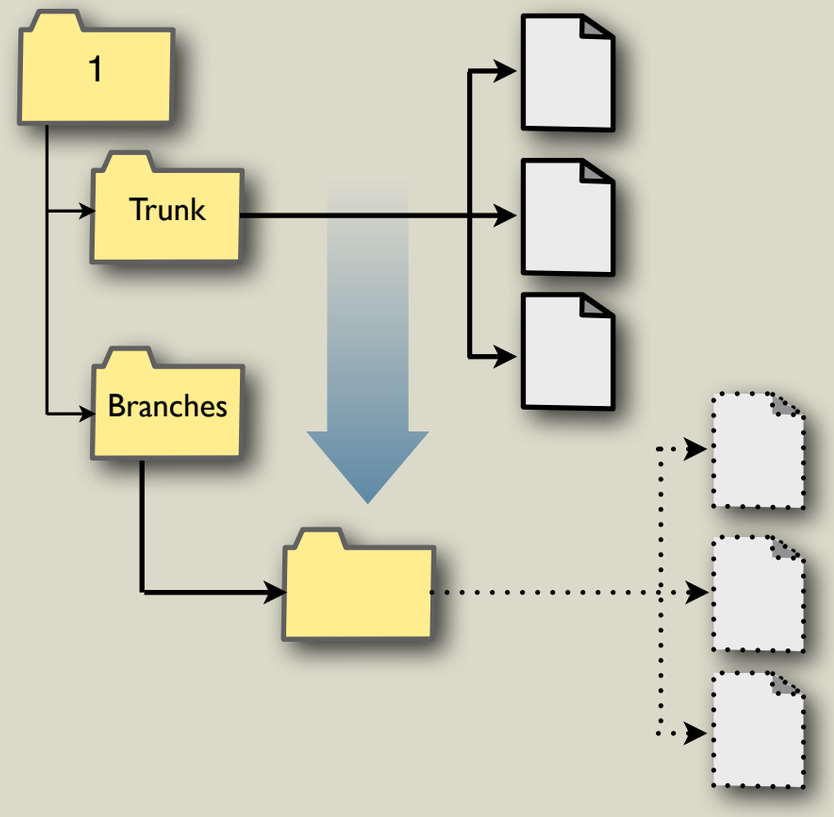
- Se satisface al cliente.
- No se perturba la linea de desarrollo actual



Bifurcaciones: creando copias



Una bifurcación es una linea de desarrollo aislada, que copia toda la información de la linea de desarrollo principal



Bifurcaciones: creando copias

Una bifurcación es una linea de desarrollo aislada, que copia toda la información de la linea de desarrollo principal

```
$ svn copy http://svn.ex.com/repos/calc/trunk \
           http://svn.ex.com/repos/calc/branches/my-calc-branch \
           -m "Creating a private branch of /calc/trunk."

Committed revision 341.

$ svn checkout http://svn.ex.com/repos/calc/branches/my-calc-branch
A  my-calc-branch/Makefile
A  my-calc-branch/integer.c
A  my-calc-branch/button.c
Checked out revision 341.
$
```

Bifurcaciones: conceptos clave

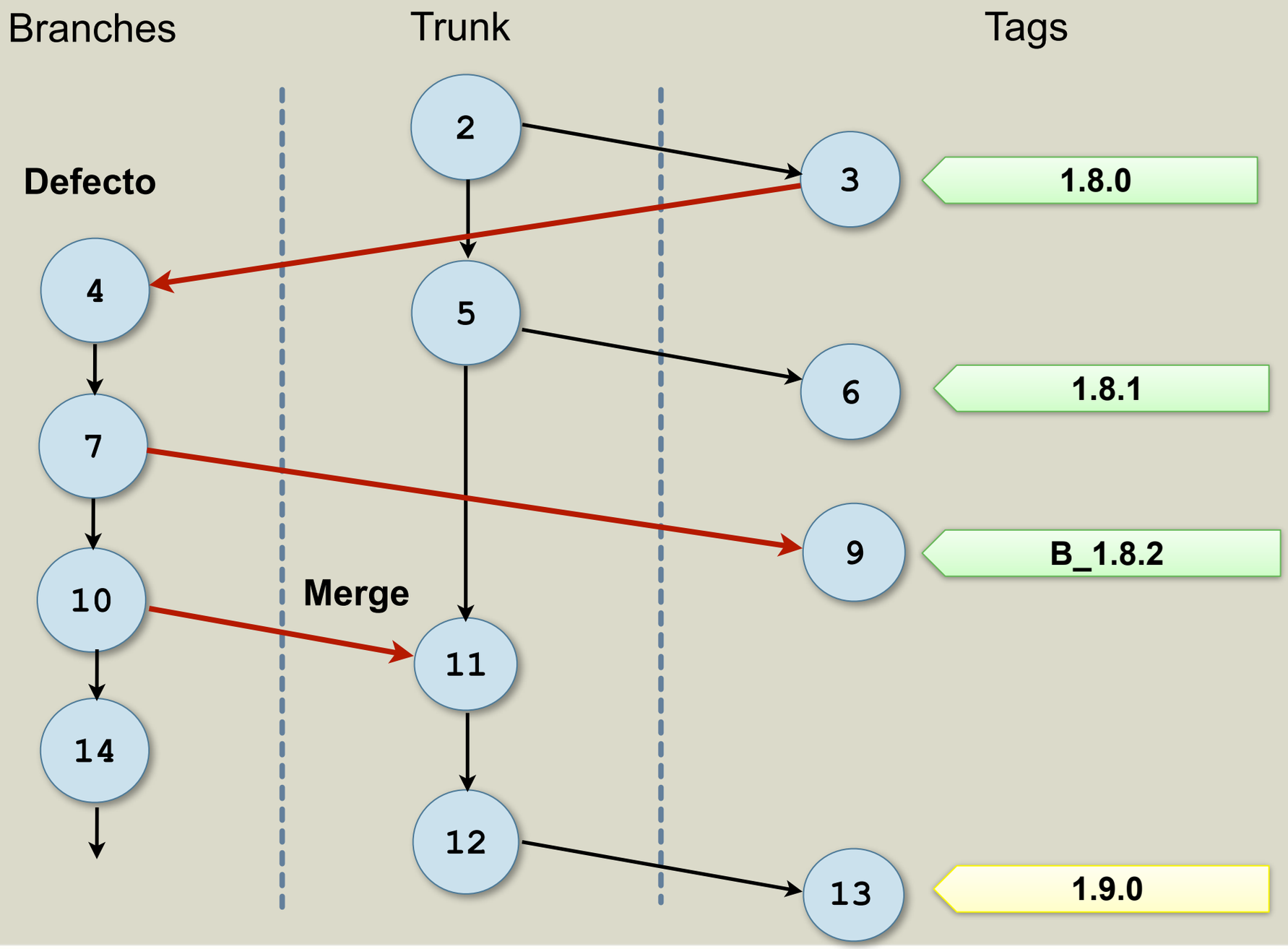
Las bifurcaciones son útiles cuando:

- se desea liberar una revisión (para producción, *testing*, etc),
- se necesita reparar un defecto,
- se quiere incorporar una nueva funcionalidad,
- se quiere realizar cambios mayores, etc.

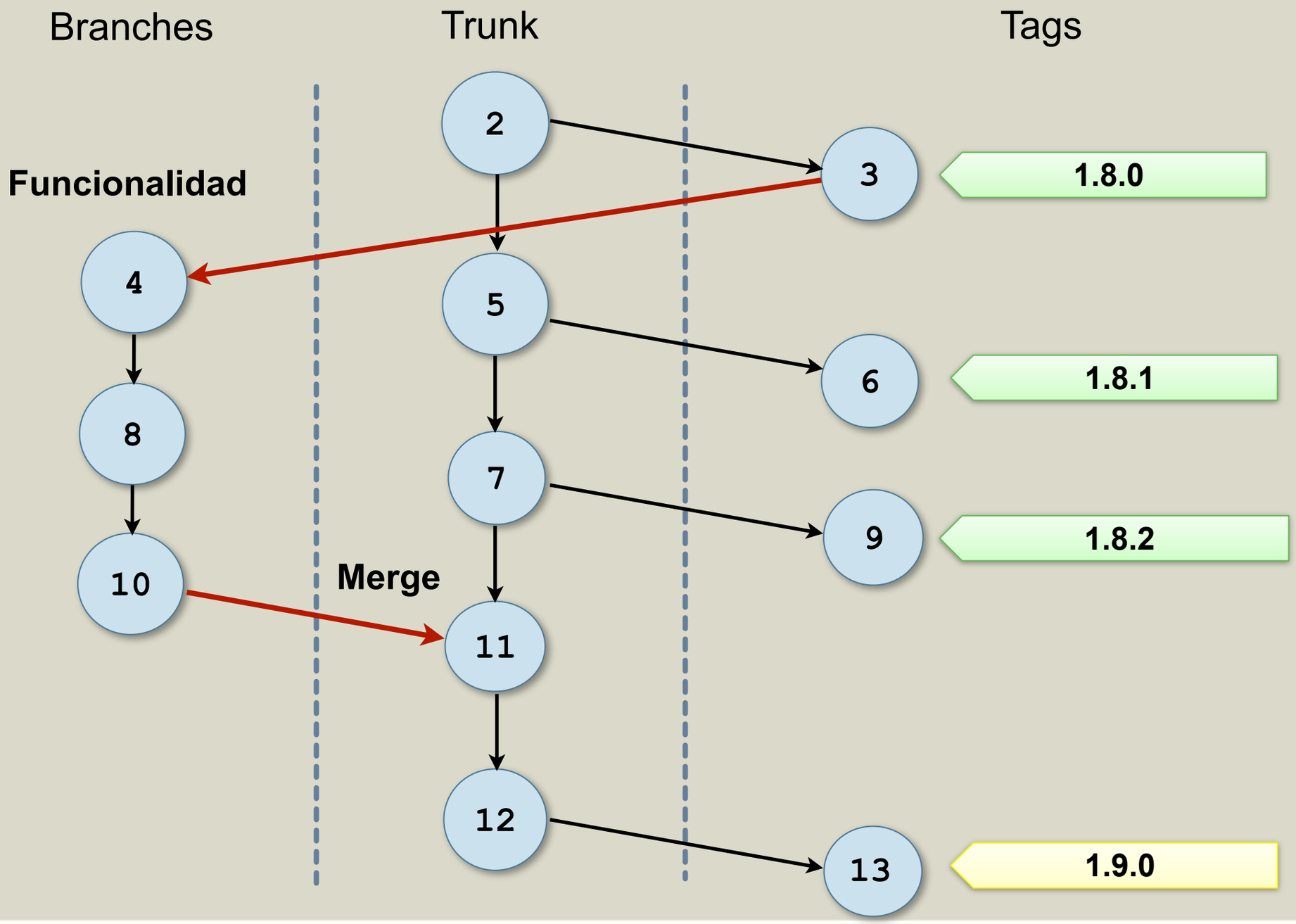
Pero **SVN no tiene un concepto interno** de *branch* o *tag*:

- Un “branch” o “tag” es sólo una copia de un repositorio.
- Esta copia *es* un *branch* (o *tag*) porque el usuario le asigna este significado.
- Esta copia es como cualquier agregado al árbol de directorios, sólo que tiene información histórica heredada.

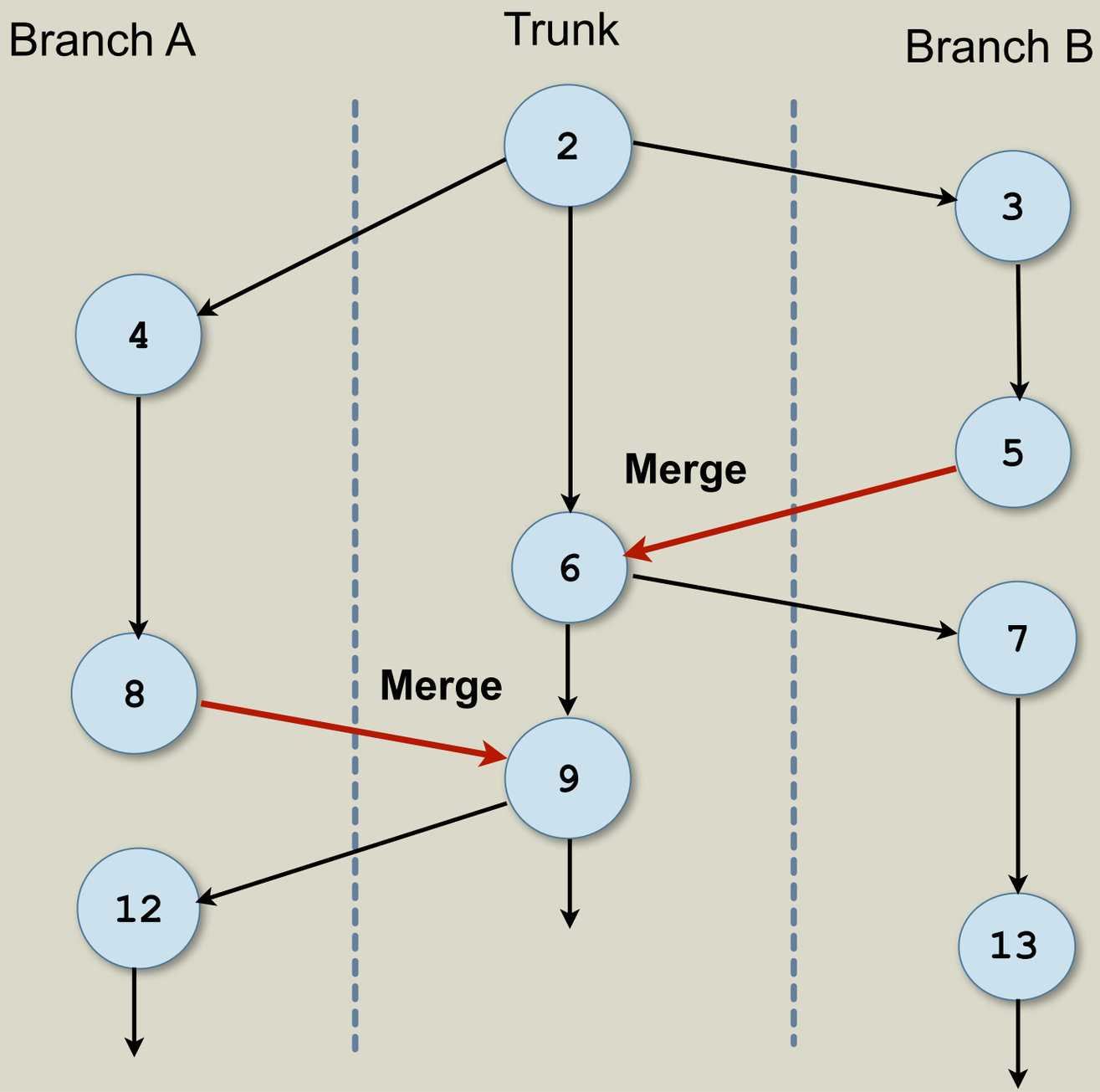
Bifurcaciones: estrategia para corrección de defectos



Bifurcaciones: estrategia por nueva funcionalidad



Bifurcaciones: estrategia por desarrollador



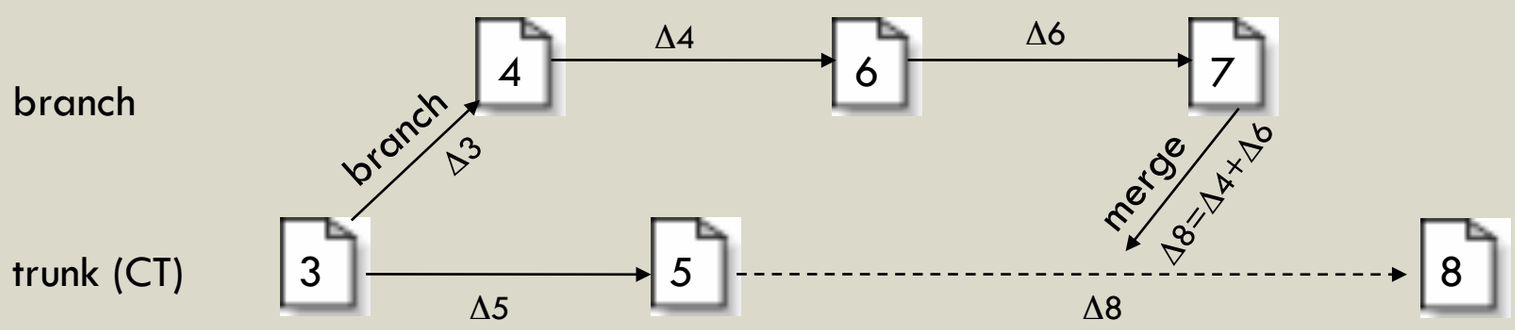
Bifurcaciones: ¿cómo volver?

• Cuando se reparar un defecto sobre una revisión de producción, normalmente se libera otra revisión (otra bifurcación).

• Pero si la bifurcación se generó por una nueva funcionalidad, o el defecto se encuentra presente en la última revisión disponible, es esperable que los cambios se integren.

• Además es deseable compartir cambios menores aún durante el proceso de desarrollo de los diferentes *branches* y la linea principal.

• Un merge es integra los cambios de una copia en otra:



Sincronización

• Normalmente mientras se desarrolla un *branch*, la linea principal sigue evolucionando.

• Es conveniente mantener sincronizado el *branch* para evaluar si los cambios son coherentes y evitar conflictos graves en el futuro.

• **Changeset**: un conjunto de cambios con un nombre único (una revisión!)

• Un *sync merge* integra todos los *changeset* del origen que no fueron previamente integrados en la copia local:

```
$ pwd
/home/user/my-calc-branch
$ svn merge ^/calc/trunk
--- Merging r345 through r356 into '.':
U    button.c
--- Recording mergeinfo for merge of r345 through r356 into '.':
 U   .
```

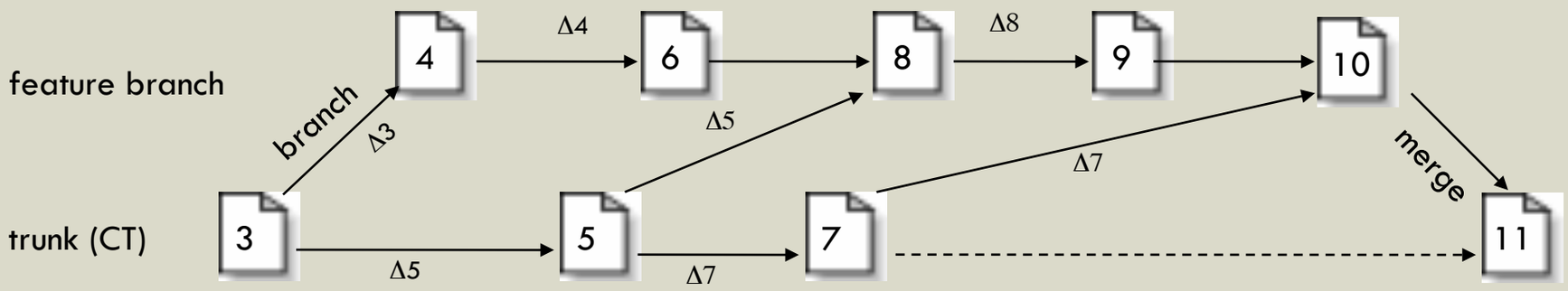
Sincronización

• Normalmente mientras se desarrolla un *branch*, la linea principal sigue evolucionando.

• Es conveniente mantener sincronizado el *branch* para evaluar si los cambios son coherentes y evitar conflictos graves en el futuro.

• **Changeset**: un conjunto de cambios con un nombre único (una revisión!)

• Un *sync merge* integra todos los *changeset* del origen que no fueron previamente integrados en la copia local:



Reintegración

- Cuando la nueva funcionalidad está lista, puede integrarse en la línea principal
- Cuando se reintegra, sólo los cambios propios del *branch* se aplican a la línea principal de desarrollo.
- Primero se sincroniza el branch con el trunk:

```
$ svn merge ^/calc/trunk
--- Merging r381 through r385 into '.':
U    button.c
U    README
--- Recording mergeinfo for merge of r381 through r385 into '.':
 U   .
$ # build, test, ...
$ svn commit -m "Final merge of trunk changes to my-calc-branch."
Sending        .
Sending        button.c
Sending        README
Transmitting file data ..
Committed revision 390.
```

Bifurcaciones y combinaciones

Reintegración

- Luego se combinan *de vuelta* todos los *changeset* del branch en el trunk:

```
$ pwd
/home/user/calc-trunk
$ svn update
Updating '.':
At revision 390.
$ svn merge --reintegrate ^/calc/branches/my-calc-branch
--- Merging differences between repository URLs into '.':
U    button.c
U    Makefile
--- Recording mergeinfo for merge between repository URLs into '.':
 U   .
$ # build, test, verify, ...
$ svn commit -m "Merge my-calc-branch back into trunk!"
Sending        .
Sending        button.c
Sending        integer.c
Sending        Makefile
Transmitting file data ..
Committed revision 391.
```

Reverse merging: deshaciendo cambios

- Un uso muy común de las combinaciones es volver para atras algún cambio:

```
$ svn merge -c -303 ^/calc/trunk
--- Reverse-merging r303 into 'integer.c':
U    integer.c
--- Recording mergeinfo for reverse merge of r303 into 'integer.c':
 U   A-branch

$ svn status
 M       .
M        integer.c

$ svn diff
…
# verify that the change is removed
…

$ svn commit -m "Undoing change committed in r303."
Sending        integer.c
Transmitting file data .
Committed revision 350.
```

Copias de revisiones específicas: reviviendo ítems

- Se puede utilizar el comando de copia para una revisión exacta de un archivo:

```
$ svn log -v
…
------------------------------------------------------------------------
r808 | joe | 2003-12-26 14:29:40 -0600 (Fri, 26 Dec 2003) | 3 lines
Changed paths:
   D /calc/trunk/real.c
   M /calc/trunk/integer.c

Added fast fourier transform functions to integer.c.
Removed real.c because code now in double.c.
…
$ svn copy ^/calc/trunk/real.c@807 ./real.c
$ svn status
A  +    real.c

$ svn commit -m "Resurrected real.c from revision 807, /calc/trunk/real.c."
Adding         real.c
Transmitting file data .
Committed revision 1390.
```

Cherrypicking

- En ocasiones es útil replicar un *changeset* de un branch en otro:

```
$ svn diff -c 355 ^/calc/trunk
Index: integer.c
===================================================================
--- integer.c(revision 354)
+++ integer.c(revision 355)
@@ -147,7 +147,7 @@
     case 8:  sprintf(info->operating_system, "Z-System"); break;
-    case 9:  sprintf(info->operating_system, "CP/MM");
+    case 9:  sprintf(info->operating_system, "CP/M"); break;
     case 10:  sprintf(info->operating_system, "TOPS-20"); break;

$ svn merge -c 355 ^/calc/trunk
--- Merging r355 into '.':
U    integer.c
--- Recording mergeinfo for merge of r355 into '.':
 U   .

$ svn status
M       integer.c
```

Bifurcaciones y combinaciones

Cherrypicking

```
$ svn propget svn:mergeinfo .
/trunk:341-349,355

$ svn mergeinfo ^/calc/trunk --show-revs eligible
r350
r351
r352
r353
r354
r356
...

$ svn merge ^/calc/trunk
--- Merging r350 through r354 into '.':
 U   .
U    integer.c
U    Makefile
--- Merging r356 through r360 into '.':
 U   .
U    integer.c
U    button.c
--- Recording mergeinfo for merge of r350 through r360 into '.':
```

Merge general

- Merge en toda su generalidad hace uso de tres argumentos principalmente: el árbol inicial, el árbol final, y la copia local que aceptará los cambios.
- Merge calcula los *changeset* necesarios para ir del árbol inicial al final y los aplica en la copia local.

```
$ svn merge http://svn.example.com/repos/branch1@150 \
            http://svn.example.com/repos/branch2@212 \
            my-working-copy

$ svn merge -r 100:200 http://svn.example.com/repos/trunk my-working-copy

$ svn merge -r 100:200 http://svn.example.com/repos/trunk
```

Combinaciones sin mergeinfo

- SVN intenta generar mergeinfo cada vez que se combinan branches para hacer mas eficiente las siguientes combinaciones. Sin embargo, en ocasiones esto no es posible:
  - Cuando se combinan arboles sin relación *histórica*.
  - Cuando se combina un repositorio externo.
  - Cuando se aplican reverse merges

- La combinacion de branches cuando se han efectuado *movimientos* también suele dar lugar a problemas.

Buenas prácticas

- Actualizar el repositorio antes de comenzar a trabajar.
- Hacer modificaciones pequeñas y aisladas, y publicarlas lo antes posible.
- Hacer la publicación archivo-por-archivo y agregar comentarios útiles.
- Resolver los conflictos manualmente.
- Evitar el modo bloqueante.
- Evitar dentro de lo posible versionar archivos binarios.
- Ser consistente con los nombres de los *tags* y *branches.*
- Mantener sincronizados los *branches* para evitar conflictos graves.
- Eliminar *tags* antiguos y *branches* de reparación de defectos cerrados.
- Antes de efectuar una operación importante (como *merge*) publicar los cambios locales para poder revertir en caso de falla.

Los 10 mandamientos de la Gestión de Configuraciones

Instituto Nacional de Tecnología Industrial
INTI

1. No crearás versiones de ningún elemento fuera del sistema de versiones (Subversion).
2. No rotularás los elementos con prefijos ni sufijos
3. Obedecerás a tu administrador de configuraciones.
4. Eliminarás lo que ya no sea usado.
5. No publicarás sin utilizar comentarios.
6. Harás referencia al tema en cuestión dentro de los comentarios de la publicación.
7. Harás bifurcaciones lo mas tarde que sea posible.
8. No incluirás código comentado dentro del sistema de versiones.
9. No publicarás sólo para hacer *backup*.
10. No publicarás ítems que rompan la compilación.

Preguntas

Instituto Nacional de Tecnología Industrial

?

Referencias

Manual de usuario de *Subversion*:

`http://svnbook.red-bean.com/nightly/en/index.html`

`http://svnbook.red-bean.com/nightly/es/index.html`

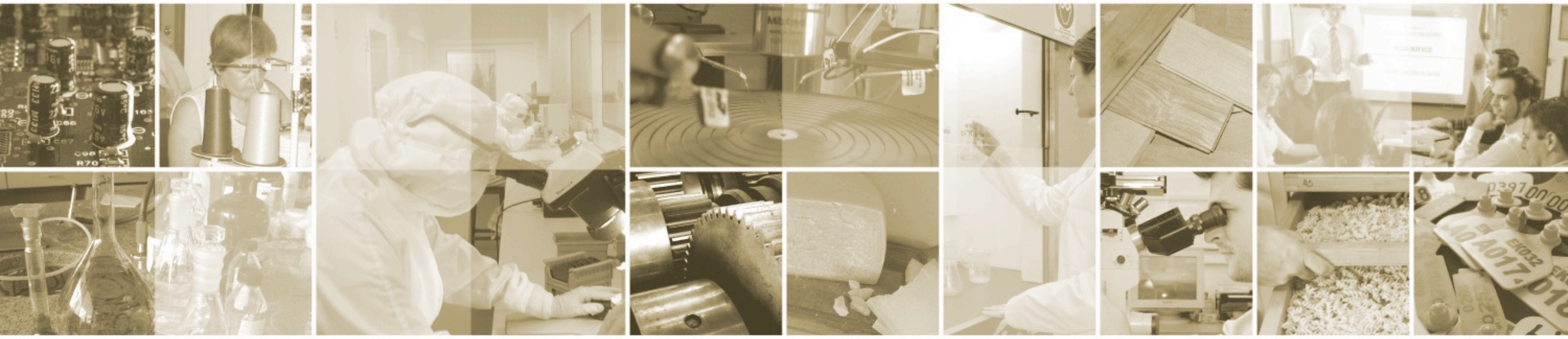
Manual de usuario de *TortoiseSVN*:

`http://tortoisesvn.net/docs/nightly/TortoiseSVN_es/index.html`

Instituto Nacional de Tecnología Industrial

INTI

Ministerio de Industria
Secretaría de industria y Comercio



Avenida Vélez Sársfield 1561
(X5000JKC) Córdoba
Córdoba, Argentina
(0351) 460 3974/468
testing@inti.gob.ar

Julio de 2011